UNIVERSIDAD DE MÁLAGA

Proyecto Final

Estándares de Datos Abiertos e Integración de Datos

Autores

Achraf Ousti El Moussati
Sebastián Rozenblum
Anabel Yu Flores Moral
Gabriela Milenova Yordanova
Karen Michell Herrera Sierra

Grado en Ingeniería de la Salud
E.T.S.I. Ingeniería Informática
Universidad de Málaga

Curso 2025/2026

Abstract

Este proyecto presenta el diseño e implementación de un ecosistema integral para la gestión, integración y explotación de datos clínico-genómicos de pacientes con melanoma acral. Partiendo de datos originales del estudio *Liang 2017 (cBioPortal)*, se ha desarrollado un flujo de trabajo modular que comienza con un proceso de extracción, transformación y carga (ETL). Este proceso se ve enriquecido dinámicamente mediante el uso de las APIs de **OncoKB** y **UniProt**. La persistencia de la información se ha realizado sobre una base de datos NoSQL (**MongoDB**), utilizando esquemas documentales anidados para modelar de forma eficiente la complejidad jerárquica de la información biomédica.

Con el objetivo de facilitar la interpretación clínica, se ha implementado un sistema de generación de reportes interactivos basado en transformaciones semánticas **XML/XSLT**. Este enfoque permite un desacoplamiento efectivo entre la persistencia y la visualización de los datos.

El núcleo del trabajo reside en la incorporación de una capa semántica fundamentada en una ontología formal (**OWL**) diseñada en Protégé. Dicha capa habilita el uso del razonador **HermiT** para la inferencia automática de estados clínicos y clasificaciones diagnósticas. Finalmente, el sistema se ha integrado en el paradigma de **Linked Data** mediante la generación de grafos RDF y la ejecución de consultas **SPARQL** avanzadas. Los resultados obtenidos validan la viabilidad de combinar bases de datos NoSQL con tecnologías de la Web Semántica para proporcionar una plataforma robusta en el ámbito de la medicina de precisión.

Palabras clave: Melanoma Acral, MongoDB, Web Semántica, OWL, SPARQL, XML, Bioinformática.

Contents

Chapter 1

Introducción

1.1 Marco teórico

En el campo de la bioinformática, la gestión eficiente y el análisis de grandes volúmenes de datos genómicos y clínicos son fundamentales para avanzar en la comprensión y el tratamiento de enfermedades como el cáncer. Los investigadores y clínicos suelen trabajar con información que proviene de múltiples fuentes: historiales clínicos de pacientes, análisis de muestras biológicas, secuenciación genómica, anotaciones moleculares y bases de datos de conocimiento oncológico. Esta diversidad de datos, junto con su naturaleza altamente interconectada y jerárquica, plantea importantes limitaciones para los sistemas de bases de datos relacionales tradicionales. Este proyecto se centra en el diseño y la implementación de una base de datos NoSQL, utilizando MongoDB, para albergar y organizar los datos del estudio *Acral Melanoma (TGEN, Genome Res 2017)*. Este conjunto de datos, disponible a través del cBioPortal for Cancer Genomics, ofrece una rica fuente de información genómica y clínica de pacientes con melanoma acral, un subtipo raro y agresivo de melanoma (cáncer de piel).

Dicho repositorio ofrece un conjunto de datos rico y representativo que incluye información clínica detallada de pacientes, características de muestras tumorales, y miles de variantes genómicas identificadas mediante secuenciación del exoma completo. Este tipo de datos biomédicos presenta características únicas que requieren soluciones tecnológicas específicas:

- **Estructura altamente anidada**: la información clínica de un paciente contiene múltiples niveles de detalle (datos demográficos, historial de tratamientos, información de seguimiento), cada uno con su propia estructura interna.
- **Relaciones complejas entre entidades**: un paciente puede tener múltiples muestras, cada muestra puede contener miles de variantes, y cada variante afecta a genes específicos con implicaciones clínicas conocidas.
- **Necesidad de enriquecimiento continuo**: los datos genómicos requieren integración con bases de datos externas (como OncoKB para significancia clínica de mutaciones, o UniProt para información proteica) que evolucionan constantemente.

- **Consultas multidimensionales**: los investigadores necesitan realizar análisis que cruzan información de pacientes, muestras y variantes de forma simultánea.

1.2 Objetivos del proyecto

Este proyecto aborda la problemática descrita mediante el diseño e implementación de un sistema integrado de gestión de datos clínicos y genómicos, aplicando tres tecnologías complementarias de estándares de datos abiertos e integración:

1. **Bases de datos NoSQL (MongoDB)** para el almacenamiento flexible y escalable de datos biomédicos jerárquicos.
2. **Tecnologías de transformación semántica (XML/XSLT)** para la generación automática de reportes clínicos visuales.
3. **Web Semántica y ontologías (OWL/SPARQL)** para la representación formal del conocimiento y consultas avanzadas.

El objetivo principal es demostrar cómo estas tecnologías, habitualmente estudiadas de forma aislada, pueden integrarse en un flujo de trabajo completo que va desde la captura de datos crudos hasta la consulta semántica del conocimiento biomédico. Específicamente, el proyecto implementa:

- Un **modelo de datos NoSQL** con tres colecciones principales (`patients`, `samples`, `variants`) más dos colecciones de enriquecimiento externo (`oncokb_genes`, `uniprot`), cada una con estructuras anidadas de hasta tres niveles que capturan la complejidad inherente a los datos clínico-genómicos.
- Un **pipeline ETL automatizado** que limpia, reestructura y enriquece los datos originales del estudio de melanoma acral, integrándolos con información actualizada de APIs públicas de relevancia oncológica.
- Un **sistema de generación de reportes** que consulta la base de datos MongoDB, transforma los resultados a XML y aplica plantillas XSLT para producir dashboards HTML interactivos que faciliten la visualización de relaciones complejas entre pacientes, muestras y variantes.
- Una **ontología OWL formal** diseñada en Protégé que modela todo el dominio del conocimiento clínico-genómico representado en la base de datos, incluyendo clases, propiedades de objeto, propiedades de datos y restricciones.
- **Capacidades de razonamiento automático** mediante reasoners como HermiT, que permiten inferir nuevo conocimiento (por ejemplo, clasificar automáticamente pacientes según su estado clínico o identificar muestras metastásicas).
- **Consultas SPARQL avanzadas** que explotan la semántica de la ontología para realizar búsquedas que serían difíciles o imposibles en bases de datos tradicionales
- **Generación automática de grafos RDF** a partir de los datos almacenados en MongoDB, creando un puente entre el almacenamiento NoSQL y la representación semántica.

1.3 Relevancia en bioinformática

La aproximación presentada en este trabajo es particularmente relevante para la bioinformática moderna por varias razones:

Gestión de heterogeneidad: Las bases de datos NoSQL permiten almacenar datos biomédicos sin forzarlos a esquemas rígidos predefinidos. Esto es crucial en investigación oncológica, donde nuevos biomarcadores, tratamientos o metodologías de secuenciación pueden requerir modificaciones frecuentes del modelo de datos.

Trazabilidad y reproducibilidad: La transformación de datos mediante tecnologías estándar como XML/XSLT garantiza que los reportes clínicos sean generados de forma determinista y auditab, esencial para la validación de resultados de investigación y para cumplir con regulaciones de datos clínicos.

Interoperabilidad semántica: El uso de ontologías OWL permite que diferentes sistemas y bases de datos biomédicas puedan compartir e integrar conocimiento de forma inequívoca. Un paciente clasificado como `PacienteFallecido` en nuestra ontología puede ser automáticamente reconocido y procesado por otros sistemas que utilicen estándares ontológicos compatibles.

Consultas basadas en conocimiento: SPARQL permite formular preguntas complejas que van más allá de la simple recuperación de datos. Por ejemplo, "encontrar todos los pacientes con variantes oncogénicas en genes asociados a resistencia terapéutica que además presentaron recurrencia de la enfermedad" es una consulta que explota tanto los datos como el conocimiento representado en la ontología.

Escalabilidad hacia medicina personalizada: La arquitectura propuesta sienta las bases para sistemas más complejos de apoyo a la decisión clínica, donde la integración de datos genómicos, clínicos y de bases de conocimiento externas es fundamental para identificar estrategias terapéuticas personalizadas.

1.4 Estructura del documento

Este documento está organizado de la siguiente manera:

- El Capítulo 1 introduce el contexto general del trabajo, exponiendo la motivación, los objetivos, la estructura del documento y las tecnologías empleadas.
- El Capítulo 2 describe la metodología empleada, detallando el diseño de la base de datos MongoDB, el pipeline ETL y de enriquecimiento, el sistema de generación de reportes mediante XML/XSLT, el modelado ontológico en Protégé, y la implementación de los scripts de generación de grafos RDF y ejecución de consultas SPARQL.
- El Capítulo 3 presenta los resultados obtenidos, incluyendo estadísticas descriptivas de los datos almacenados, ejemplos de reportes HTML generados, la ontología resultante con sus inferencias, y los resultados de las consultas SPARQL sobre el grafo RDF.

- El Capítulo 4 discute las conclusiones del trabajo, reflexiona sobre las limitaciones encontradas, y propone líneas de trabajo futuro para extender este sistema hacia aplicaciones clínicas reales y su integración con otras fuentes de datos biomédicos.
- Finalmente, el Capítulo 5 expone las conclusiones del trabajo, incluyendo los aprendizajes alcanzados, las limitaciones enfrentadas y posibles direcciones a seguir en trabajos futuros dentro de esta línea.

Chapter 2

Literature Review

En este capítulo se describe el marco metodológico y técnico empleado para el desarrollo del sistema integral de gestión de datos clínico-genómicos. El flujo de trabajo se ha diseñado siguiendo un enfoque modular que abarca desde la persistencia de datos en sistemas NoSQL hasta la representación avanzada del conocimiento mediante tecnologías de la Web Semántica.

2.1 Diseño y estructuración de la base de datos NoSQL

La fase inicial del proyecto consistió en el diseño de una base de datos NoSQL utilizando **MongoDB**. La elección de este sistema se fundamenta en su capacidad para gestionar la heterogeneidad de los datos biomédicos sin las restricciones de un esquema rígido. El diseño se estructuró en torno a tres colecciones principales interconectadas: `patients`, `samples` y `variants`.

Cada una de estas colecciones fue diseñada para cumplir con un mínimo de tres niveles de anidamiento jerárquico. Esto permite representar de forma natural relaciones complejas, como la localización genómica de las variantes o el historial detallado de tratamientos y seguimiento clínico. La integridad de la información se mantiene mediante un sistema de referencias cruzadas basado en identificadores únicos, asegurando la trazabilidad total desde el perfil demográfico del paciente hasta la mutación a nivel de nucleótido.

2.2 Pipeline ETL y proceso de enriquecimiento

Para transformar los datos crudos procedentes de cBioPortal en documentos estructurados, se desarrolló un pipeline de extracción, transformación y carga (ETL) implementado en el script `conversion_mongobd.py`. Este flujo operativo comienza con una etapa de limpieza donde se normalizan las cadenas de texto y se gestionan los valores nulos para garantizar la consistencia del repositorio.

Posteriormente, el script ejecuta funciones de reestructuración profunda que convierten las tablas planas en formato CSV hacia documentos JSON de alta complejidad jerárquica. Un aspecto fundamental de esta fase es el enriquecimiento dinámico en tiempo real. Mediante el consumo de las APIs

de **OncoKB** y **UniProt**, el sistema recupera automáticamente información sobre la significancia oncológica de los genes y las funciones proteicas asociadas, integrando esta información externa directamente en las colecciones de soporte.

2.3 Arquitectura de reportes clínicos (XML/XSLT)

Con el fin de facilitar la interpretación de los datos, se implementó un sistema genérico de generación de reportes basado en el script `mongoxml_to_html.py`. Esta herramienta actúa como un motor de transformación semántica que desacopla la lógica de persistencia de la capa de presentación.

El proceso se inicia con la ejecución de consultas de agregación en MongoDB, definidas externamente en el archivo `queries.txt`. Los resultados obtenidos se procesan mediante la librería `lxml` para generar un documento intermedio en formato XML que preserva la estructura anidada original. Finalmente, se aplica una hoja de estilos `template.xslt` para transformar el XML en una interfaz HTML interactiva, incorporando funcionalidades como el colapsado dinámico de variantes para mejorar la usabilidad del reporte clínico.

2.4 Modelado ontológico y representación del conocimiento

Para alcanzar la interoperabilidad semántica, se diseñó la ontología formal `melanoma_es` utilizando el editor Protégé. Este modelo organiza el dominio mediante una red de clases y propiedades que definen las interacciones biológicas y clínicas fundamentales.

La ontología incorpora lógica descriptiva avanzada para permitir el uso del razonador **HermiT**. Se configuraron reglas de equivalencia para la clasificación automática de individuos, lo que permite que el sistema infiera de forma autónoma el estado clínico de los pacientes o la naturaleza de las muestras tumorales. Este proceso asegura que el conocimiento extraído sea consistente con los estándares internacionales de la Web Semántica.

2.5 Implementación de grafos RDF y motor SPARQL

La fase final traslada los datos NoSQL hacia el paradigma de los datos enlazados. Mediante el script `reto5.py`, se automatizó la conversión de los documentos JSON en un grafo de tripletas RDF, asignando identificadores únicos (URIs) a cada recurso.

Para la explotación de este grafo, se implementó el script `reto6.py`, diseñado como un motor genérico de ejecución de consultas SPARQL. Este script es capaz de cargar dinámicamente el grafo generado y procesar consultas complejas de agregación y filtrado. Esto permite realizar análisis bioinformáticos que atraviesan todos los niveles jerárquicos de la ontología, extrayendo tanto la información explícita como el conocimiento inferido por el razonador lógico.

Chapter 3

Methodology

Para estructurar la información de manera coherente y facilitar consultas complejas, se ha optado por un diseño que consta de **tres colecciones interconectadas**: `patients`, `samples` y `variants`. Esta estructura nos permitirá no solo capturar la información de cada paciente de forma individual, sino también trazar las relaciones entre los pacientes, las muestras biológicas obtenidas de ellos y las variantes genómicas identificadas en dichas muestras.

A continuación, se detalla la estructura propuesta para cada una de las colecciones:

3.0.1 Colección: `patients`

Esta colección almacena la información demográfica y clínica de cada paciente incluido en el estudio.

- **Nivel 1:** Información básica del paciente.
  - `patient_id`: Identificador único del paciente.
  - `sex`: Sexo del paciente.
  - `race_category`: Categoría racial del paciente.
  - `age_at_diagnosis`: Edad del paciente en el momento del diagnóstico.
- **Nivel 2:** Historial clínico y de tratamiento.
  - `clinical_history`: Objeto con información sobre el historial médico del paciente.
    - `initial_diagnosis_date`: Fecha del diagnóstico inicial.
    - `primary_tumor_site`: Localización del tumor primario.
  - `treatments`: Array de objetos que detalla los tratamientos recibidos.
    - `treatment_type`: Tipo de tratamiento (e.g., "Ipilimumab", "Interferon").
    - `start_date`: Fecha de inicio del tratamiento.
    - `end_date`: Fecha de finalización del tratamiento.

- **Nivel 3:** Seguimiento y estado de la enfermedad.
  - `follow_up`: Objeto con información de seguimiento.
    - `disease_free_months`: Meses libre de enfermedad.
    - `disease_free_status`: Estado de la enfermedad (e.g., "0:DiseaseFree", "1:Recurred/Progressed").

3.0.2 Colección: `samples`

Contendrá información detallada sobre cada muestra biológica extraída de los pacientes.

- **Nivel 1:** Identificación y tipo de muestra.
  - `sample_id`: Identificador único de la muestra.
  - `patient_id`: Identificador del paciente al que pertenece la muestra (referencia a la colección `patients`).
  - `sample_type`: Tipo de muestra (e.g., "Primary", "Metastasis").
- **Nivel 2:** Detalles de la recolección y procesamiento.
  - `collection_info`: Objeto con detalles de la recolección.
    - `collection_date`: Fecha de recolección de la muestra.
    - `collection_method`: Método de recolección.
  - `processing_info`: Objeto con información del procesamiento.
    - `processing_date`: Fecha de procesamiento.
    - `sequencing_type`: Tipo de secuenciación realizada (e.g., "Whole Exome Sequencing").
- **Nivel 3:** Datos de análisis molecular.
  - `molecular_data`: Objeto que alberga datos moleculares.
    - `mutation_count`: Número de mutaciones identificadas.
    - `copy_number_alterations`: Array de objetos con información sobre alteraciones en el número de copias.

3.0.3 Colección: `variants`

Esta colección albergará la información específica de cada variante genómica identificada en las muestras.

- **Nivel 1:** Identificación de la variante.
  - `variant_id`: Identificador único de la variante.

  - `sample_id`: Identificador de la muestra en la que se encontró la variante (referencia a la colección `samples`).
  - `gene_symbol`: Símbolo del gen afectado.
- **Nivel 2:** Características de la variante.
  - `variant_details`: Objeto con las características de la variante.
    * `chromosome`: Cromosoma donde se localiza la variante.
    * `start_position`: Posición de inicio de la variante.
    * `end_position`: Posición de finalización de la variante.
    * `reference_allele`: Alelo de referencia.
    * `alternate_allele`: Alelo alternativo.
- **Nivel 3:** Anotación funcional y predicciones.
  - `functional_annotation`: Objeto con la anotación funcional.
    * `variant_classification`: Clasificación de la variante (e.g., "Missense_Mutation", "Nonsense_Mutation").
    * `protein_change`: Cambio en la proteína resultante.
    * `sift_prediction`: Predicción del impacto de la variante por SIFT.
    * `polyphen_prediction`: Predicción del impacto de la variante por PolyPhen.

3.1 Sistema de generación automática de reportes (Tarea 2)

se desarrolló un sistema genérico para la generación automática de vistas HTML a partir de consultas realizadas sobre la base de datos MongoDB. Este sistema permite integrar tecnologías NoSQL con estándares de transformación documental, conectando MongoDB con XML, XSLT y HTML dentro de un mismo flujo de trabajo. El objetivo principal es permitir que cualquier consulta definida por el usuario pueda transformarse automáticamente en una vista visual sin necesidad de modificar el código del sistema.

Diseño de las consultas MongoDB

Para la Tarea 2 se diseñaron tres consultas sobre MongoDB empleando el framework de agregación (`aggregate`) con el objetivo de obtener documentos JSON con suficiente riqueza y anidamiento como para: (i) explotar las relaciones definidas entre colecciones en la Tarea 1 y (ii) generar posteriormente vistas HTML significativas mediante transformación a XML y XSLT. Las consultas se definieron en un fichero externo `queries.txt`, de forma que el sistema pudiera ejecutarlas sin necesidad de modificar el código Python, manteniendo así un comportamiento genérico y reutilizable.

Desde un punto de vista de diseño, las tres consultas cubren tres niveles de exploración: una vista centrada en el paciente (pacientes–muestras–variantes), una vista centrada en la variante enriquecida con conocimiento externo (OncoKB) y contexto clínico (paciente y muestra), y una vista "completa" orientada a inspección (variant–sample–patient). En todas ellas se aplican operadores de agregación como `$lookup`, `$unwind`, `$project` y filtros por expresiones (`$expr`) para realizar uniones equivalentes a *joins* relacionales, pero preservando la estructura jerárquica propia de MongoDB.

Consulta 1: `patients_samples_variants` **(vista integrada por paciente)** Esta consulta parte de la colección `patients` y construye un documento agregado que integra la información clínica y de supervivencia del paciente junto con sus muestras y variantes asociadas. El flujo principal es:

- **Limitación de resultados** (`$limit: 20`) para controlar el tamaño de salida y facilitar pruebas y generación rápida de HTML.
- **Unión patients → samples** mediante `$lookup` usando como clave `patient_id` (en `patients`) y `patient.id` (en `samples`). El resultado se incorpora como un array `samples`.
- **Unión samples → variants** usando `$lookup` con `pipeline` y variables (`let`). Se crea la variable `sample_ids` con la lista `$samples.sample_id` y se filtran variantes con `$expr` y `$in`.
- **Selección de campos** mediante `$project`, devolviendo `patient_id`, `clinical`, `survival`, `recurrence`, `samples` y un resumen de `variants`.

Esta consulta genera una vista integrada de historia clínica con información clínica y genómica combinada.

Consulta 2: `variants_with_oncokb` **(enriquecimiento externo)** La segunda consulta se centra en la colección `variants` y combina información genómica con conocimiento externo de `oncokb_genes` y el contexto clínico:

- **Limitación** (`$limit: 20`).
- **Unión variants → oncokb_genes** con `$lookup` usando `gene.symbol`.
- **Unión variants → samples** usando `samples.tumor_sample` → `sample_id`.
- **Unión samples → patients** usando `sample.patient.id` → `patient_id`.
- **Normalización** con `$unwind`.
- **Proyección final** usando `$arrayElemAt` para extraer valores de `oncokb`.

Esta consulta transforma datos crudos en datos interpretados con valor biológico.

Consulta 3: `variants_full_info` **(vista completa)** Esta consulta genera una vista completa que conecta cada variante con su muestra y paciente:

- **Limitación** (`$limit: 20`).
- **Unión variants → samples** con `$lookup`.

- **Normalización** con `$unwind: "$sample"`.
- **Unión samples → patients** con `$lookup`.
- **Segundo** `$unwind` para obtener un único documento por variante.

Consideraciones de diseño

- Uso de `$limit` para pruebas controladas.
- Empleo de `$project` para reducir tamaño de salida.
- Uso de `$lookup` con `pipeline` para relaciones basadas en listas.
- Recomendación de indexar `patient_id`, `sample_id` y `samples.tumor_sample`.

Transformación de JSON a XML

Una vez ejecutada cada consulta sobre MongoDB, el resultado se obtiene como una lista de documentos JSON (Python `list` de `dict`) que conserva la estructura anidada original del modelo de datos. En este proyecto, los documentos devueltos pueden contener múltiples niveles de anidamiento y heterogeneidad estructural: diccionarios (objetos), listas (arrays) y valores primitivos, además de campos con nombres no directamente compatibles con XML (por ejemplo, `_id` o claves con puntos). Por ello, se implementó una conversión **genérica, recursiva y robusta** que permite transformar *cualquier* salida JSON en un árbol XML válido y navegable, sin asumir un esquema fijo.

Normalización de claves para XML MongoDB permite nombres de campos que no siempre son válidos como etiquetas XML. Para resolverlo, el script implementa una función de normalización (`normalize_key`) que:

- Convierte el campo especial `_id` en `id`, evitando el carácter `_` inicial como etiqueta principal y facilitando su lectura en XSLT.
- Sustituye el carácter punto `.` por guion bajo `_` (p.ej., `patient.id` → `patient_id`), ya que las etiquetas XML no permiten puntos en el nombre.
- Sustituye el símbolo `$` por el prefijo `DOLLAR_`, evitando colisiones con sintaxis propia de MongoDB.

Esta normalización permite que el XML resultante sea **válido**, consistente y transformable mediante XSLT sin necesidad de lógica adicional para tratar casos especiales.

Conversión recursiva de estructuras JSON La transformación se realiza mediante dos funciones: `json_to_xml` (crea la raíz) y `build_xml` (recorre recursivamente la estructura). La idea principal es mapear cada tipo JSON a una representación XML explícita:

- **Diccionarios (objetos):** se representan como un nodo con atributo `type="object"` y se crea un subelemento por cada par clave–valor. Cada clave se normaliza y se convierte en una etiqueta XML.

- **Listas (arrays):** se representan como un nodo con atributo `type="list"` y cada elemento se encapsula en una etiqueta `element`. Esto unifica el tratamiento de listas aunque contengan objetos complejos o valores simples.
- **Valores primitivos (string, int, float, boolean, null):** se serializan como texto dentro del nodo. En el caso de `null`, se escribe una cadena vacía para evitar errores de transformación.

Este enfoque es especialmente útil en bases de datos NoSQL, donde el esquema puede variar entre documentos y donde pueden existir arrays de objetos con niveles profundos (por ejemplo, un paciente que contiene múltiples muestras, y cada muestra variantes, y cada variante anotaciones y metadatos). La recursividad garantiza que la conversión funcione correctamente independientemente de la profundidad del anidamiento.

Estructura XML resultante El XML generado se construye siempre bajo una raíz `<root>`, y dado que las consultas devuelven una lista de documentos, el nivel inmediatamente inferior suele ser un conjunto de `<element>` (uno por documento). A modo ilustrativo, la salida sigue el patrón:

```
<root type="list">
<element type="object">
<patient_id>...</patient_id>
<clinical type="object">...</clinical>
<samples type="list">
<element type="object">...</element>
</samples>
</element>
</root>
```

La inclusión del atributo `type` (con valores `object` o `list`) aporta semántica adicional que facilita el diseño de plantillas XSLT, ya que permite distinguir explícitamente entre nodos que representan objetos y nodos que representan colecciones.

Serialización y persistencia del XML Finalmente, el árbol XML se serializa a disco mediante `lxml.etree.ElementTree.write` con `pretty_print=true` y codificación `UTF-8`. Para cada consulta se genera un fichero independiente con nombre `<consulta>.xml`, permitiendo depuración y trazabilidad del flujo:

- Comprobar que la consulta devuelve datos antes de aplicar XSLT.
- Validar que la estructura XML coincide con lo esperado por la plantilla.
- Reutilizar el XML como salida intermedia interoperable.

En conjunto, esta etapa asegura una transformación fiable desde JSON (flexible y anidado) hacia XML (estructurado y estándar), habilitando la fase posterior de generación de HTML mediante XSLT.

Aplicación de XSLT y generación de HTML

Una vez obtenido el documento XML de salida para cada consulta, el siguiente paso consiste en transformar dicho XML en una vista HTML legible y reutilizable mediante una hoja de estilo XSLT. En esta tarea se optó por XSLT 1.0 debido a su amplia compatibilidad y a su facilidad para definir transformaciones declarativas basadas en patrones. La plantilla `template.xslt` está diseñada con un enfoque **genérico**, de manera que pueda aplicarse a distintos resultados de consulta siempre que el XML de entrada mantenga la estructura común generada por el script (raíz `root` con elementos `element` y atributo `type` para diferenciar objetos y listas).

Estrategia general de visualización La plantilla implementa una estrategia de renderizado basada en tablas HTML anidadas. A nivel superior, el resultado completo se presenta como una tabla donde:

- Las **columnas** se generan dinámicamente a partir de las claves del primer registro (`root/element[1]/*`), evitando codificar nombres de campos fijos.
- Las **filas** se generan recorriendo cada documento de salida (`root/element`).

Este diseño es especialmente útil en MongoDB, donde las consultas pueden devolver estructuras con distintos campos o subdocumentos dependiendo de la etapa del pipeline. De esta forma, la plantilla se adapta automáticamente a la salida sin necesidad de reescribir HTML para cada consulta.

Uso del atributo `type` para distinguir objetos y listas La conversión JSON→XML introdujo el atributo `type` con valores `object` y `list`. Esta decisión simplifica significativamente el XSLT, permitiendo aplicar renderizado condicional con `xsl:choose`:

- Si `@type='object'`, el contenido se representa como una tabla interna de pares clave–valor.
- Si `@type='list'`, el contenido se renderiza como una lista de elementos, que puede ser a su vez una lista de objetos o una lista de valores simples.
- Si no existe `@type`, el contenido se interpreta como valor simple (texto) y se inserta directamente en la celda HTML.

Gracias a esta semántica, la plantilla puede recorrer estructuras anidadas de forma robusta, incluso cuando aparecen objetos dentro de objetos y listas dentro de objetos.

Plantillas reutilizables: `renderObject` y `renderList` Para evitar duplicación de lógica, la transformación se apoya en dos plantillas nombradas:

- **`renderObject`:** recibe un nodo que representa un objeto (`type="object"`) y genera una tabla anidada donde cada fila corresponde a una clave y su valor. En cada valor se vuelve a aplicar una lógica recursiva: si el valor es otro objeto se vuelve a invocar `renderObject`; si es una lista se invoca `renderList`; y si es simple se imprime el texto.

- `renderList:` recibe un nodo lista (`type="list"`) y diferencia tres casos: (i) lista de objetos (cuando `element/*` existe), (ii) lista de valores simples (cuando `element` no tiene subnodos) y (iii) un caso especial para listas de variantes.

Esta separación permite que el HTML final mantenga una estructura coherente: objetos como tablas clave–valor y listas como colecciones de bloques repetidos, favoreciendo la legibilidad.

Caso especial: visualización colapsable de variantes Debido a la naturaleza del dominio genómico, la colección de variantes puede ser voluminosa incluso tras aplicar limitaciones (`$limit`) y proyecciones. Para evitar generar páginas HTML excesivamente largas, la plantilla define un **caso especial** para el nodo `variants` dentro de `renderList`. En este caso:

- Se construye un identificador único con `generate-id($node)` para cada bloque de variantes.
- Se muestra un **resumen** con el número de variantes (`../variants_count`) junto a un botón "Mostrar / Ocultar".
- Las variantes se renderizan en un contenedor HTML inicialmente oculto (`style="display:none"`), que se puede desplegar bajo demanda.

Este comportamiento se implementa con un pequeño fragmento de JavaScript embebido (`toggleVisibility(id)`), que alterna la visibilidad de cada bloque. El objetivo es mejorar la usabilidad del reporte y reducir la sobrecarga visual cuando existen múltiples variantes asociadas a un mismo paciente o muestra.

Estilos y legibilidad del reporte La plantilla incluye estilos CSS básicos (tipografía, márgenes, bordes y colores de cabecera) para presentar la información de forma clara. Se utiliza `border-collapse` y un sombreado ligero en `th` para diferenciar encabezados de datos, y la clase `nested-table` permite ajustar el espaciado de tablas internas, manteniendo la jerarquía visual.

Salida HTML y desacoplamiento del diseño El uso de XSLT permite desacoplar el **diseño** del **proceso de consulta** y transformación. Así, es posible:

- modificar la presentación (colores, estructura, agrupaciones) sin tocar el script Python,
- reutilizar el mismo XSLT para varias consultas,
- o definir plantillas específicas por consulta manteniendo el mismo XML intermedio.

En resumen, la etapa XSLT convierte resultados complejos provenientes de MongoDB en vistas HTML estructuradas y navegables, manteniendo un enfoque genérico y reutilizable, y añadiendo mecanismos de interacción (colapsado) para mejorar la exploración de listas voluminosas como las variantes genéticas.

Script genérico

El script desarrollado para esta tarea ha sido diseñado con un enfoque **genérico y reutilizable**, de forma que pueda aplicarse no solo a la base de datos de melanoma utilizada en el proyecto,

sino también a cualquier otra base de datos MongoDB con estructura jerárquica compatible. Para ello, el programa no contiene nombres de colecciones ni campos codificados de manera fija, sino que interpreta dinámicamente la información proporcionada por el usuario en tiempo de ejecución.

La entrada del script se define completamente mediante argumentos por línea de comandos, que permiten especificar:

- La **URI de conexión** a MongoDB.
- El **nombre de la base de datos**.
- Un **archivo de texto** con la definición de las consultas en formato MongoDB (find o aggregate).
- Un **documento XSLT** con la plantilla de visualización.
- El **directorio de salida** para los ficheros generados.

Gracias a esta parametrización, el mismo script puede reutilizarse para distintas bases de datos, distintas consultas y distintos formatos de presentación, sin necesidad de modificar el código fuente.

Asimismo, el programa es capaz de procesar consultas de distinta complejidad, incluyendo pipelines de agregación con `$lookup`, `$project`, `$unwind` y otras etapas propias de MongoDB. Esto permite explotar relaciones entre colecciones y generar vistas enriquecidas sin acoplar el script a un esquema concreto.

En definitiva, el carácter genérico del script convierte esta herramienta en un componente flexible dentro del flujo de trabajo, permitiendo su reutilización en otros proyectos que requieran transformar resultados MongoDB a XML y posteriormente a HTML mediante XSLT.

Flujo completo

La Figura 3.1 muestra el flujo completo del proceso implementado en la Tarea 2, desde la entrada de credenciales y consultas hasta la generación final del documento HTML mediante XML y XSLT.

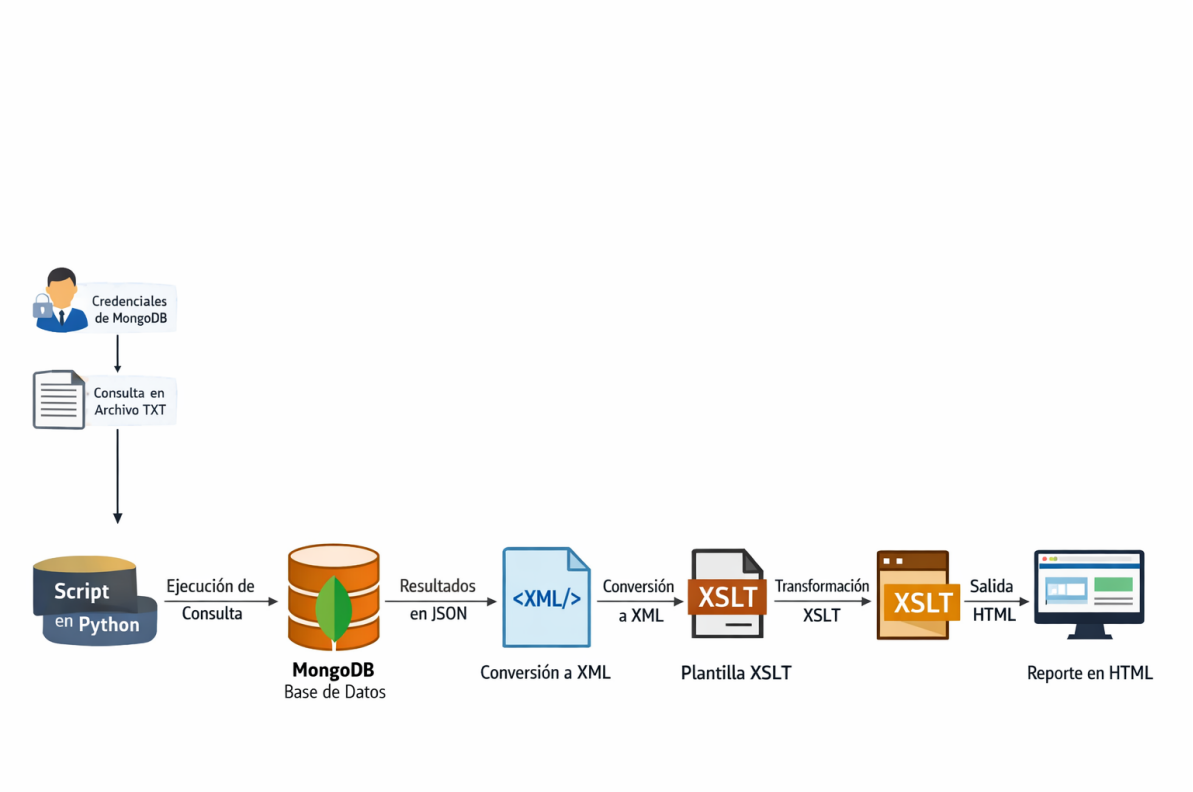

Figure 3.1: Flujo completo del proceso MongoDB → XML → XSLT → HTML.

3.2 Diseño de la Ontología y Modelado Semántico (Tarea 3)

Tras la consolidación de la base de datos NoSQL, se procedió al diseño de una ontología formal que capturase la semántica del dominio. El modelado se realizó en Protégé, estructurando el conocimiento en una jerarquía de clases que reflejan fielmente las entidades de la base de datos: `Paciente`, `Muestra`, `Variante`, `Gen` y `Proteina`. Para articular las interacciones entre estas clases, se definieron diversas *Object Properties* (propiedades de objeto) que establecen los vínculos biológicos y clínicos, tales como la relación entre una variante y el gen al que afecta (`afectaAGen`) o la pertenencia de una muestra a un paciente determinado (`perteneceAPaciente`). Asimismo, se incorporaron *Data Properties* (propiedades de datos) para integrar los literales y valores métricos, mapeando campos como la edad del diagnóstico, el símbolo del gen o las predicciones funcionales de impacto.

3.2.1 Poblado, Inferencia y Razonamiento

El poblado de la ontología se abordó de manera dual. En una primera fase, se introdujeron individuos de forma manual en Protégé para validar la consistencia lógica del modelo y asegurar que las consultas SPARQL posteriores tuviesen un entorno de pruebas controlado. Una vez estructurado el grafo, se aplicó el razonador **HermiT**, cuya función principal es mejorar la accesibilidad de la información mediante la inferencia de tipos. Se configuraron reglas de lógica descriptiva para clasificar automáticamente a los individuos basándose en sus atributos; esto permite, por ejemplo, que el sistema identifique de forma autónoma estados clínicos críticos o tipos de muestras específicas,

optimizando así la recuperación de información en las fases de consulta.

3.2.2 Estrategia de Consultas SPARQL y Cobertura del Espacio de Búsqueda

Con el objetivo de explotar el grafo de conocimiento, se diseñaron **seis consultas SPARQL** orientadas a cubrir la totalidad del espacio de búsqueda de la ontología. El diseño de estas consultas no fue aleatorio, sino que se estructuró para validar diferentes niveles de complejidad: desde la exploración de metadatos (clases y propiedades instanciadas) hasta la extracción de conocimiento inferido por el razonador. Se incluyeron consultas de **agregación** y **filtrado** semántico que permiten, por ejemplo, agrupar variantes por su impacto funcional o contabilizar alteraciones genómicas navegando a través de las relaciones entre variantes y genes, demostrando una capacidad de análisis superior a la que ofrecen los esquemas relacionales o documentales tradicionales.

3.2.3 Automatización y Scripts Genéricos (Reto)

Para escalar el sistema a un entorno de producción, se implementaron dos herramientas en Python utilizando la librería **RDFlib**. El primer script, `reto5.py`, automatiza la generación del grafo de tripletas RDF a partir de los documentos JSON de MongoDB, garantizando que cada registro sea transformado en un recurso semántico con su URI correspondiente. El segundo componente, `reto6.py`, se diseñó como un motor de ejecución de consultas con un carácter estrictamente genérico. Mediante la parametrización de la entrada, este script es capaz de cargar el grafo generado y ejecutar el archivo de consultas externas, permitiendo que el sistema sea reutilizable para cualquier ontología o conjunto de datos que siga un esquema compatible. Este enfoque desacopla la lógica de ejecución de la definición de las consultas, facilitando el mantenimiento y la escalabilidad del proyecto.

3.2.4 Resultados de la Automatización y Ejecución Genérica (Reto)

Uno de los hitos técnicos del proyecto fue la transición desatendida de los datos desde un entorno NoSQL hacia un entorno semántico. En esta sección se validan los resultados de los scripts de automatización.

Generación del Grafo mediante RDFlib (`reto5.py`)

Tras la ejecución del script `reto5.py`, se obtuvo un archivo en formato Turtle (`.ttl`) que representa el grafo completo de la base de datos. El script logró mapear exitosamente los documentos JSON complejos hacia una estructura de tripletas. Se verificó que:

- Se generaron URIs únicas y persistentes para cada paciente, muestra y variante.
- Las colecciones anidadas de MongoDB fueron "aplanadas" correctamente en relaciones semánticas.
- El archivo resultante es totalmente compatible con herramientas externas de validación RDF.

Ejecución Genérica de Consultas (`reto6.py`)

La validación final consistió en lanzar las seis consultas SPARQL mediante el script `reto6.py`. A diferencia de una ejecución manual en Protégé, este script demostró un comportamiento genérico:

1. **Flexibilidad:** El programa cargó el grafo dinámicamente y procesó un archivo de texto con las seis consultas sin requerir cambios en el código fuente.
2. **Exportación de Resultados:** Por cada consulta ejecutada, el script generó automáticamente un archivo CSV en la carpeta de resultados. Esto facilita el análisis posterior de los datos por parte de bioinformáticos que utilicen herramientas de estadística como R o Python (Pandas).

Este flujo de trabajo automatizado (MongoDB $\rightarrow$ RDF $\rightarrow$ SPARQL $\rightarrow$ CSV) representa una solución escalable para la integración de datos abiertos en oncología, permitiendo que nuevas muestras añadidas a la base de datos sean analizadas semánticamente en cuestión de segundos.

Chapter 4

Results

4.1 Sistema de generación automática de reportes (Tarea 2)

En esta sección se muestran los resultados obtenidos tras la ejecución del script genérico desarrollado en la Tarea 2. El sistema procesa consultas MongoDB, genera documentos XML intermedios y aplica una plantilla XSLT para producir vistas HTML finales.

La ejecución se realizó mediante el siguiente comando:

```
python mongoxml_to_html.py \
--uri <URI_MONGO> \
--db EstadaresProyecto \
--queries queries.txt \
--xslt template.xslt \
--outdir resultados/mongo_a_html
```

Resultados XML

Tras ejecutar el script, se generan automáticamente documentos XML correspondientes a cada consulta definida en el archivo `queries.txt`. La Figura 4.1 muestra un ejemplo del XML generado.

Figure 4.1: Ejemplo de documento XML generado a partir de una consulta MongoDB.

Resultados HTML

Posteriormente, cada documento XML es transformado mediante la plantilla XSLT, obteniéndose un archivo HTML con una visualización estructurada y navegable. La Figura 4.2 muestra un ejemplo del resultado final.

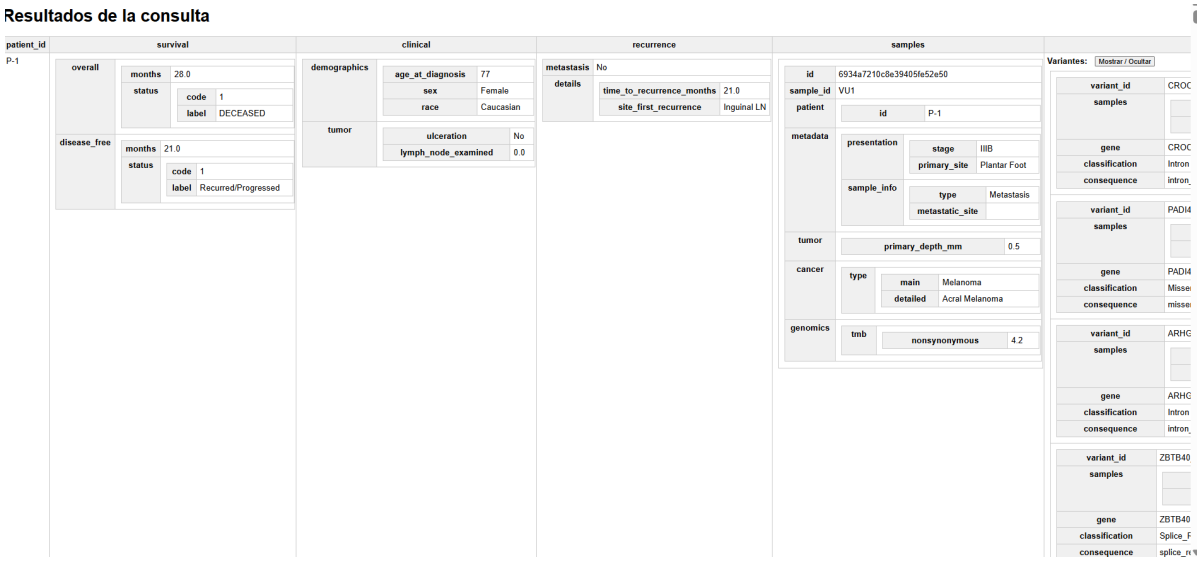

Figure 4.2: Vista HTML generada automáticamente a partir del XML mediante XSLT.

El sistema se probó con tres consultas diferentes, todas ellas con estructuras de salida distintas. En todos los casos se generaron correctamente los documentos XML y HTML, demostrando la capacidad genérica del script para adaptarse a distintas consultas sin necesidad de modificar el código.

4.2 Estructura de la Ontología y Jerarquía de Clases (Tarea 3)

Como resultado del modelado semántico, se ha obtenido una jerarquía de clases sólida que refleja todas las entidades del dominio de melanoma acral (Figura 4.3). En la imagen se puede apreciar la organización de las clases raíz bajo `owl:Thing`, destacando la creación de clases definidas o equivalentes, identificadas por el icono de las tres líneas horizontales.

Clases como `MuestraMetastasis` (subclase de `Muestra`) y `PacienteFallecido` (subclase de `Paciente`) han sido configuradas mediante lógica descriptiva. Esto permite que el sistema no solo almacene datos, sino que sea capaz de categorizar individuos automáticamente en función de sus propiedades diagnósticas y de supervivencia, cumpliendo así con el objetivo de mejorar la accesibilidad y la semántica de la información clínica.

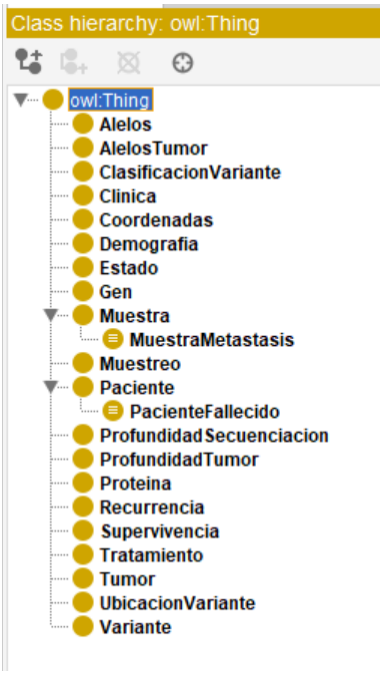

Figure 4.3: Jerarquía de clases de la ontología `melanoma_es` en Protégé, mostrando la estructura jerárquica y las clases definidas para el razonamiento automático.

4.2.1 Análisis de resultados mediante consultas SPARQL

Una vez generado el grafo de conocimiento en formato Turtle (`.ttl`) y validado mediante el razonador, se procedió a la explotación de los datos mediante el lenguaje de consulta SPARQL. El objetivo de estas consultas es demostrar la capacidad del sistema para integrar información jerárquica y semántica que sería costosa de obtener mediante lenguajes de consulta tradicionales.

Exploración de la topología y atributos del grafo

Las consultas 3 y 4 se centraron en validar la estructura del grafo generado a partir de la base de datos NoSQL. La **Consulta 3** permite identificar todas las propiedades de objeto (*Object Properties*) que han sido instanciadas, lo que verifica que las relaciones definidas en la ontología (como `afectaAGen` o `tieneMuestra`) se han mapeado correctamente desde MongoDB. Por otro lado, la **Consulta 4** extrae los valores literales (*Data Properties*), asegurando que los identificadores, fechas y métricas clínicas son accesibles y mantienen su tipado correcto.

Validación estructural e inventario de recursos (Consultas 1 a 4)

Antes de proceder al análisis biológico, se ejecutaron cuatro consultas orientadas a validar la integridad del grafo generado por el script `reto5.py`.

La **Consulta 1** y la **Consulta 2** permitieron verificar el mapeo de clases. Mientras que la primera listó todas las clases presentes (`mel:Paciente`, `mel:Variante`, etc.), la segunda filtró exclusivamente los individuos con nombre, asegurando que no existieran nodos huérfanos o "blank nodes" inesperados que dificultaran la trazabilidad.

Por otro lado, las **Consultas 3 y 4** se diseñaron para auditar el contenido del grafo. La Consulta 3 extrajo todas las *Object Properties* activas (como `mel:afectaAGen`), confirmando que la red de relaciones entre entidades clínicas y genómicas se mantenía fiel al diseño original de MongoDB. Finalmente, la Consulta 4 recuperó los valores literales (*Data Properties*), validando que tipos de datos sensibles como fechas (`xsd:date`) y recuentos genómicos (`xsd:integer`) se importaron con el tipado correcto. Este paso previo de auditoría fue esencial para garantizar que los resultados de las consultas posteriores (5 y 6) fueran biológicamente coherentes.

Validación del razonamiento semántico

La **Consulta 5** es crítica para evaluar el éxito de la integración semántica. Su objetivo es listar los tipos o clases a los que pertenece cada individuo tras la ejecución del razonador OWL.

```
# ------------------------------------------------------------
# Consulta 5 { Tipos inferidos tras razonamiento OWL
# ------------------------------------------------------------
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?individuo ?claseInferida
WHERE {
  ?individuo rdf:type ?claseInferida .
  FILTER (
    ?claseInferida != owl:NamedIndividual &&
    ?claseInferida != owl:Thing
  )
}
ORDER BY ?claseInferida
```

Listing 1: Consulta para extraer clases inferidas por el razonador.

Los resultados de esta consulta confirmaron que el sistema clasifica automáticamente a los pacientes según su estado clínico. Individuos que en la base de datos original solo poseían el atributo "DECEASED" aparecen aquí clasificados bajo la clase `PacienteFallecido`, demostrando que el grafo de conocimiento es capaz de interpretar la lógica médica definida en la ontología.

Análisis genómico: Variantes por gen (Consulta 6)

La consulta final representa la utilidad práctica del sistema para un bioinformático. Esta consulta realiza un recuento de las variantes genéticas identificadas, agrupándolas por el símbolo del gen afectado. Para ello, la consulta debe navegar a través de la relación semántica entre la variante y la entidad genómica correspondiente.

```
# ----------------------------------------------------------
# Consulta 6 { Número de variantes genéticas por gen
# ----------------------------------------------------------
PREFIX mel: <http://example.org/melanoma_es#>

SELECT ?geneSymbol (COUNT(?variante) AS ?numVariantes)
WHERE {
  ?variante a mel:variants ;
            mel:gene ?vg .
  ?vg mel:symbol ?geneSymbol .
}
GROUP BY ?geneSymbol
ORDER BY DESC(?numVariantes)
```

Listing 2: Consulta SPARQL de agregación para el conteo de variantes por gen.

Interpretación de resultados: Esta consulta revela la distribución de mutaciones en la cohorte de melanoma acral estudiada. Los resultados obtenidos muestran, por ejemplo, una alta incidencia de variantes en genes como `ACAP3`, `NLRC5` y `MUTYH`.

La importancia de este resultado radica en que SPARQL permite realizar este conteo de forma directa sobre el grafo de conocimiento, sin necesidad de realizar múltiples *joins* manuales como ocurriría en SQL o procesos de filtrado complejos en scripts externos. Al estar los datos "enlazados", el sistema entiende que una variante "pertenece" a un gen y puede realizar la agregación semántica de forma nativa. Esto facilita la identificación de genes frecuentemente mutados (*drivers*) en el estudio, proporcionando una herramienta de análisis poderosa para la investigación oncológica.

Chapter 5

Conclusiones y Trabajo Futuro

5.1 Logros alcanzados

En el marco de este proyecto, se ha logrado consolidar una infraestructura integral para la gestión y explotación de datos clínico-genómicos centrada en el melanoma acral. La implementación ha permitido unificar información inicialmente fragmentada de cBioPortal y enriquecerla mediante la conexión con bases de conocimiento de referencia como **OncoKB** y **UniProt**. El resultado final es un sistema que trasciende el simple almacenamiento, transformando datos brutos en conocimiento estructurado y procesable.

5.2 Valoración técnica del sistema

Desde una perspectiva técnica, el empleo del paradigma documental a través de **MongoDB** ha demostrado ser una elección idónea. La capacidad de gestionar la naturaleza jerárquica de los datos biomédicos sin las restricciones de los esquemas relacionales rígidos facilita enormemente la escalabilidad del sistema. El diseño anidado de las colecciones garantiza una alta eficiencia, permitiendo que la historia clínica y las alteraciones genómicas coexistan en un modelo coherente.

Por otro lado, el uso de transformaciones **XML/XSLT** dota al sistema de una gran versatilidad. El desacoplamiento logrado entre la persistencia y la presentación permite generar visualizaciones clínicas dinámicas sin comprometer la integridad estructural del repositorio.

5.3 Impacto de la capa semántica

El avance más significativo de esta propuesta reside en la incorporación de la capa semántica basada en el estándar **OWL**. La transición hacia el ecosistema de **Linked Data** permite que el sistema realice razonamiento automático mediante el razonador **HermiT**. Esta capacidad de inferencia es fundamental para descubrir estados clínicos y clasificaciones biológicas que no se encuentran explícitas en las fuentes originales, situando a esta plataforma como una herramienta potente para la medicina personalizada y la investigación oncológica avanzada.

5.4 Limitaciones y líneas futuras

A pesar de la solidez de los resultados, se han identificado áreas de mejora para futuras etapas de desarrollo:

- **Enfoque multiómico:** Sería de gran valor incorporar datos de expresión génica (RNA-Seq) para obtener una visión más sistémica del tumor.
- **Apertura de datos:** El desarrollo de un *endpoint* SPARQL público facilitaría la federación de consultas con otros repositorios internacionales.
- **Automatización avanzada:** La implementación de herramientas de mapeo automático como RML permitiría que el grafo RDF se actualice de forma desatendida conforme evolucione la base de datos MongoDB.

Estas líneas futuras aseguran que el proyecto no sea una solución estática, sino una base modular preparada para el apoyo continuo a la investigación biomédica.

Chapter 6

Bibliography

[1] Liang, W. S., et al. (2017). *Integrated genomic analyses of acral melanoma*. Genome Research, 27(10), 1625-1638. doi:10.1101/gr.223859.117.

[2] Gao, J., et al. (2013). *Integrative analysis of complex cancer genomics and clinical profiles using the cBioPortal*. Science Signaling, 6(269), pl1.

[3] MongoDB, Inc. *MongoDB Documentation: The Document Database*. Disponible en: `https://www.mongodb.com/docs/`.

[4] W3C. *Resource Description Framework (RDF) Concepts and Abstract Syntax*. Disponible en: `https://www.w3.org/TR/rdf11-concepts/`.

[5] W3C. *OWL 2 Web Ontology Language Document Overview*. Disponible en: `https://www.w3.org/TR/owl2-overview/`.

[6] W3C. *SPARQL 1.1 Overview*. Disponible en: `https://www.w3.org/TR/sparql11-overview/`.

[7] Team RDFLib. *RDFLib v7.0.0 documentation*. Disponible en: `https://rdflib.readthedocs.io/`.

[8] Behnel, S., et al. *lxml - XML and HTML with Python*. Disponible en: `https://lxml.de/`.