UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE

“INTEGRACIÓN DE BASES DE DATOS HETEROGÉNEAS UTILIZANDO GRAPHQL”

CAMILO PARRA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Carlos Buil
Profesor Correferente: Cecilia Reyes

Enero - 2022

DEDICATORIA

Este trabajo esta principalmente dedicado a mi familia que me ha apoyado en todo momento, que me han dado las oportunidades y los medios para poder seguir creciendo, y a mis amigos, con los que he compartido toda esta vida universitaria y quienes han hecho de esta experiencia una mucho más divertida.

RESUMEN

Resumen— La integración de datos heterogéneos es un problema en sí mismo que trae consigo la tarea de proveer una vista unificada de múltiples fuentes de datos. El tener fuentes de datos heterogéneas trae consigo los desafíos de identificar y comprender el cómo manejan los datos y cómo están estructurados. En este trabajo se diseñó un modelo y esquema para manejar los datos provenientes de 5 bases de datos heterogéneas, utilizando el lenguaje GraphQL para proveer integración a las consultas sobre los datos, las cuales se realizarán sobre la API que implemente este esquema, entregando además los resultados obtenidos de pruebas realizadas con los distintos tipos de consultas que ofrece esta.

Palabras Clave— Integración de datos heterogéneos, GraphQL, Sistema de integración, Bases de datos heterogéneas

ABSTRACT

Abstract— Heterogeneous data integration is a problem in itself that brings with it the task of providing a unified view of multiple data sources. Having heterogeneous data sources brings with it the challenges of identifying and understanding how the data is handled and structured. In this work is designed a model and schema to handle data from 5 heterogeneous databases, using the GraphQL language to provide integration to data queries, which will be performed on the API that implements this schema, also delivering the results obtained from tests performed with different types of queries offered by this.

Keywords— Heterogeneous Data Integration, GraphQL, Integration System, Heterogeneous Databases.

GLOSARIO

ADN: Ácido Desoxirribonucleico.

AGP: A Golden Path.

API: *Application Programming Interface*. Interfaz de programación de aplicaciones.

dbGaP: *Database of Genotypes and Phenotypes*.. Base de datos de Genotipos y Fenotipos.

dbVar: *Database of Genomic Structural Variation*. Base de datos de variación estructural genómica.

DDBJ: *DNA Database of Japan*. Base de datios de DNA de Japón

DML: *Data Manipulation Language*. Lenguaje de manipulación de datos.

DQL: *Data Querying Language*. Lenguaje de consulta de datos.

ENA: *European Molecular Biology Laboratory* .Laboratorio Europeo de Biología Molecular.

EMBL-Bank: *European Molecular Biology Laboratory Nucleotide Sequence Database*. Base de datos de secuencias de nucleótidos del Laboratorio Europeo de Biología Molecular.

EST: *Expressed Sequence Tags*. Etiquetas de secuencias expresadas

FTP: *File Transfer Protocol*. Protocolo de transferencia de archivos.

GAV: *Global as View*.

GEO: *Gene Expression Omnibus*. Omnibus de Expresión Génica.

GLAV: *Global local As View*.

GSS: *Genome Survey Sequences*. Secuencias de estudio del genoma.

HTML: *HyperText Markup Language*. Lenguaje de marcado de hypertexto.

HTTP: *Hypertext Transfer Protocol*. Protocolo de transferencia de hipertexto.

INSDC: *International Nucleotide Sequence Database Consortium*.

Consorcio Internacional de Bases de datos de secuencias de nucleótidos.

ISSN: *International Standard Serial Number*. Número de serie estándar internacional.

ISO: *International Organization for Standardization*. Organización Internacional de Normalización.

JSON: *JavaScript Object Notation*. Notación de objeto de JavaScript.

LAV: *Local As View*.

MEDLINE: *Medical Literature Analysis and Retrieval System Online*. Sistema de análisis y recuperación de la literatura médica en línea. MeSH: *Medical Subject Headings*. MIAMI: *Minimum information about a microarray experiment*. Información mínima sobre un experimento de microarrays.

NCBI: *National Center for Biotechnology Information*.

NIH: *National Institutes of Health*. Institutos Nacionales de Salud.

NLM: *National Library of Medicine*. Biblioteca Nacional de Medicina de los Estados Unidos.

OWL: *Web ontology language*. Lenguaje de ontología web.

PMC: PubMed Central.

PDB: *Protein Data Bank*. Banco de datos de proteinas.

RefSeq: *Reference Sequence*. Secuencia de referencia.

REST: *Representational State Transfer*.Transferencia de Estado Representacional.

RDF: *Resource Description Framework*.Marco de Descripción de Recursos.

SDL: *Schema Definition Language*. Lenguaje de definición de esquema.

dbSNP: *Database of Short Genetic Variations*. Base de datos de variaciones genéticas cortas.

SOA: *ServiceOriented Architecture*.Arquitectura orientada a servicios.

SOAP: *Simple Object Access Protocol*. Protocolo de acceso a objetos simples.

SRA: *Sequence Read Archive*. Archivo de lectura de secuencias.

TPA: *Third Party Annotation*. Anotaciones de Terceros. TSA: *Transcriptome Shotgun Assembly*. Ensamblaje de escopeta de transcriptoma.

UID: *Unique Identifier*. Identificador Único. URL: *Uniform Resource Locator*. Localizador de recursos uniforme.

WGS: *Whole Genome Shotgun*. Escopeta de genoma completo.

WSDL: *Web Services Description Language*. Lenguaje de descripción de servicios web.

WWW: *World Wide Web*. Red informática mundial.

XML: *Extensible Markup Language*. Lenguaje de Marcado Extensible.

ÍNDICE DE CONTENIDOS

RESUMEN III

ABSTRACT III

GLOSARIO IV

ÍNDICE DE FIGURAS IX

ÍNDICE DE TABLAS XI

INTRODUCCIÓN 1

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA 3
- 1.1 NCBI y sus bases de datos biológicas 3
  - 1.1.1 Base de datos *Genomes* 4
- 1.2 Trabajo relacionado 6
  - 1.2.1 Integración basada en ontologías 6
  - 1.2.2 Otros Acercamientos 7
- 1.3 Objetivos 8
  - 1.3.1 Objetivo General 8
  - 1.3.2 Objetivos Específicos 8

CAPÍTULO 2: MARCO CONCEPTUAL 9
- 2.1 API 9
  - 2.1.1 Servicios webs 9
  - 2.1.2 REST 10
  - 2.1.3 *Service Oriented Architectures* 11
- 2.2 GraphQL 12
  - 2.2.1 Manejo de recursos 14
  - 2.2.2 Esquema 15
  - 2.2.3 *Resolvers* 15
  - 2.2.4 Obtención de datos 16
  - 2.2.5 Manejo de errores y permisos 17
- 2.3 NCBI 18
  - 2.3.1 Entrez 19
  - 2.3.2 *Assembly* 20
  - 2.3.3 *Bioproject* 21
  - 2.3.4 *BioSample* 22
  - 2.3.5 PubMed 23
  - 2.3.6 *Nucleotide* 23

CAPÍTULO 3: PROPUESTA DE SOLUCIÓN 25
3.1 Metodología de trabajo 25
3.2 Toma, documentación y verificación de requisitos 25
3.2.1 Datos 25
3.2.2 Consultas 26
3.3 Análisis y Diseño 27
3.3.1 Análisis y extracción de datos genómicos 27
3.3.2 Estructura de un archivo de un genoma 28
3.3.3 Definición de base de datos y modelo inicial 31
3.3.4 Esquema de GraphQL y mejora en el modelo de base de datos 36
3.3.5 Arquitectura de la solución 38
3.4 Implementación de la solucion 39
3.4.1 Selección de datos iniciales 40
3.4.2 Obtención de archivos de genomas mediante la API de Entrez 41
3.4.3 Inicialización de la base de datos 44
3.4.4 Creación de modelos y esquemas 47
3.4.5 Creación de servicio y los *resolvers* de GraphQL 54
3.4.6 Obtención de nuevos genomas 57

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN 58
4.1 Resultados de consultas para documento grande 59
4.2 Resultados de consultas para documento de tamaño mediano 62
4.3 Resultados de consultas para documento pequeño 65
4.4 Tiempo de procesamiento de genomas nuevos 68

CAPÍTULO 5: CONCLUSIONES 69
5.1 Objetivo general 69
5.2 Objetivos específicos 69
5.2.1 Diseño del esquema que integrará los datos 69
5.2.2 Diseño de un mecanismo para abstraer y mapear los datos heterogéneos 69
5.2.3 Implementar una API utilizando el esquema diseñado 70
5.2.4 Evaluar el rendimiento de la solución implementada 70
5.3 Limitaciones 70
5.4 Trabajo futuro 71
5.5 Palabras finales 72

ANEXOS 73
5.1 Listado de bases de datos de la NCBI 73

REFERENCIAS BIBLIOGRÁFICAS 77

ÍNDICE DE FIGURAS

1 Búsqueda global utilizando el buscador de la página web de la NCBI. 4

2 Ejemplo de una estructura básica de una API. 9

3 Ejemplo de una *Query* y su resultado en el lenguaje GraphQL. 12

4 Ejemplo de un esquema en GraphQL. 13

5 Arquitectura simple de una API REST y una API de GraphQL. 14

6 Ejemplo de una multiples llamadas en una arquitectura REST. 15

7 Ejemplo de una llamada utilizando el esquema de GraphQL. 16

8 Ejemplo de una respuesta con datos parciales en GraphQL. 18

9 Representación visual de un genoma bacteriano realizada mediante CGview y editado manualmente mediante Adobe illustrator. 24

10 Archivo de texto plano correspondiente a una secuencia de un genoma. 28

11 Archivo de secuencia de un genoma, sección de referencias. 29

12 Archivo de secuencia de un genoma, sección de Features. 31

13 Tiempo de carga y ejecucón para 100.000 registros con 5 bases de datos. . . . 33

14 Ejemplo de documento en MongoDB. 33

15 Modelo inicial para almacenar genomas. 34

16 Modelo con *features* almacenados como subdocumentos embebidos. 34

17 Modelo con *features* almacenados en otra colección y utilizando referencias. . 35

18 Esquema de GraphQL para genomas. 36

19 Esquema de GraphQL para *features*. 36

20 Modelo y esquema de genoma que incluye referencias a otras bases de datos. . 37

21 Arquitectura inicial de la API de GraphQL. 38

22 Arquitectura de la API con caché y conexión a la base de datos *Genome*. 39

23 Archivo de texto assembly_summary_refseq. 40

24 Arquitectura de la API actualizada con la base de datos *Nucleotide*. 42

25 Resultado de una consulta utilizando la herramienta ESearch de Entrez. 43

26 Pseudocódigo del algoritmo de extracción y almacenamiento de datos de genomas. 45

27 Modelo de genoma que incluye el *Geninfo Identifier*(gi) y nuevas referencias utilizando los identificadores que maneja Entrez 46

28 Definición de clase Genoma utilizando decoradores de Typegoose y TypeGraphQL. 48

29 Definición del esquema Genoma con los campos adicionales para la integración de datos. 49

30 Esquema de GraphQL para la información proveniente de la base de datos *Assembly*. 50

31 Esquema de GraphQL para la información proveniente de la base de datos *BioProject*. 53

32 Esquema de GraphQL para la información proveniente de la base de datos *BioSample*. 54

33 Esquema de GraphQL para la información proveniente de la base de datos *Pubmed*. 54

34 Pseudocódigo del algoritmo de obtención y almacenamiento de un nuevo genoma. 57

35 Tiempos de respuesta para genoma NC_003282 buscando por *genome accession*. 59

36 Tiempos de respuesta para genoma NC_003282 utilizando caché y buscando por *genome accession*. 59

37 Tiempos de respuesta para genoma NC_003282 buscando por *assembly accession*. 60

38 Tiempos de respuesta para genoma NC_003282 utilizando caché y buscando por *assembly accession*. 60

39 Tiempos de respuesta para genoma NC_003282 buscando por *locus tag*. 61

40 Tiempos de respuesta para genoma NC_003282 utilizando caché y buscando por *locus tag*. 61

41 Tiempos de respuesta para genoma NT_037436 buscando por *genome accession*. 62

42 Tiempos de respuesta para genoma NT_037436 utilizando caché y buscando por *genome accession*. 62

43 Tiempos de respuesta para genoma NT_037436 buscando por *assembly accession*. 63

44 Tiempos de respuesta para genoma NT_037436 utilizando caché y buscando por *assembly accession*. 63

45 Tiempos de respuesta para genoma NT_037436 buscando por *locus tag*. 64

46 Tiempos de respuesta para genoma NT_037436 utilizando caché y buscando por *locus tag*. 64

47 Tiempos de respuesta para genoma NW_007931121 buscando por *genome accession*. 65

48 Tiempos de respuesta para genoma NW_007931121 utilizando caché y buscando por *genome accession*. 65

49 Tiempos de respuesta para genoma NW_007931121 buscando por *assembly accession*. 66

50 Tiempos de respuesta para genoma NW_007931121 utilizando caché y buscando por *assembly accession*. 66

51 Tiempos de respuesta para genoma NW_007931121 buscando por *locus tag*. . . 67

52 Tiempos de respuesta para genoma NW_007931121 utilizando caché y buscando por *locus tag*. 67

53 Tiempos de procesamiento para distintos tamaños de archivos de genomas. . . 68

ÍNDICE DE TABLAS

1 Ejemplo de listado de archivos en el servidor FTP para genoma GCA_000001215. 5

2 Bases de datos de la NCBI que maneja Entrez. 73

INTRODUCCIÓN

En la actualidad, la gran cantidad de fuentes de datos heterogéneos que van creciendo en tamaño, complejidad y variedad además de la creciente demanda por el acceso a estas trae consigo la necesidad de mecanismos eficientes de integración para manipular esta información distribuida. La heterogeneidad de los datos provenientes de distintas fuentes, especialmente datos biológicos, es un bien conocido obstáculo que limita la integración y análisis de los datos, ya que, si los mismos datos son expresados de distintas maneras, entender y compartir estos datos se convierte en un proceso que requiere una labor intensiva y propensa a errores [Khoumbati *et al.*, 2005].

Integrar estas fuentes de datos distintas trae consigo desafíos que se necesitan considerar como lo son:

- La forma en que cada fuente de datos maneja las consultas y manipulaciones mediante sus propios lenguajes de consulta y manipulación de datos (DQL y DML).
- El cómo opera cada fuente de datos con su propio modelo, como lo son los modelos relacionales, no estructurados, etc, siendo necesaria una forma de mapear las relaciones entre las entidades existentes para integrarlas en una vista unificada.
- Mantener la consistencia entre las fuentes de datos y el sistema unificado que integrará estos datos, ya que pueden presentarse representaciones irregulares de los datos o la pérdida de estos, cosas que deben ser manejadas para evitar inconsistencias.
- Los distintos niveles de abstracción que tiene cada fuente de datos, como por ejemplo la diferencia en la forma de manejar las marcas de tiempo, mientras una fuente puede utilizar días y meses, otra puede usar horas, días y meses, por lo que se debe mantener la consistencia en el modelo que unifica estas dos fuentes, siendo una de las prácticas más comunes elegir la que tenga un mayor nivel de abstracción, en este caso días y meses.
- Las diferencias semánticas entre las fuentes de datos, en donde los significados de ciertos nombres campos, tablas y datos son parecidos, pero no significan lo mismo, junto con la diferencia entre nombres que se refieren a un mismo objeto son una de las tareas que consumen más tiempo al momento de integrar los esquemas de las fuentes.

Existen distintas técnicas que pueden ser llevadas a cabo para realizar una integración de datos, como lo son las técnicas lógicas [Levy y Minker, 2000], basadas en ontologías [Gagnon, 2007] , técnicas que utilizan inteligencia artificial [Levy, 1999] y una de las más prevalentes, los mediadores [Wiederhold, 1992].

Los mediadores proveen una abstracción entre las fuentes de datos y cómo es accedida y representada dentro de un sistema de información, típicamente por medio de un esquema, o en algunos casos, modelos de datos, los cuales son creados de distintas maneras y van evolucionando a través del tiempo.

En el presente trabajo se planteará y construirá una solución para integrar la base de datos de genomas perteneciente al National Center for Biotechnology Information (NCBI) que contiene información pública relacionada con los genomas y genes de distintas especies, con información de proyectos, muestras y literatura asociada a estos genomas, presentes en diversas bases de datos disponibilizadas por el NCBI, mediante la utilización de GraphQL y su flexible esquema que permite integrar distintas fuentes de datos, a la vez de optimizar las consultas realizadas dependiendo de qué información se requiere en cada una de estas.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

La integración de datos heterogéneos ha sido un tema que gradualmente ha ido tomando importancia debido a la gran cantidad de datos que se va generando y que a la vez se va guardando de formas distintas. Por lo mismo es que se hace difícil la tarea de integrar datos con los distintos tipos de diferencias que existen entre ellos, ya sea en los nombres, la semántica o en el contenido. Es por esto que se buscan integrar estos datos mediante la creación de modelos de datos globales que funcionen como una abstracción que represente el contenido de varios modelos de datos heterogéneos, ya sea mediante la traducción de datos, manual o automática, guardando los datos en un formato estándar, pudiendo ser accedidos de forma uniforme, o la traducción de las *Querys* que trabaja con consultas directas a las fuentes de datos heterogéneas. En este trabajo se busca integrar cinco bases de datos biológicos heterogéneos desarrollando una API que permita consultar los datos integrados, los cuales serán utilizados por un software de visualización que busca identificar contextos genómicos y visualizar genomas bacterianos, siendo necesaria información de estas bases de datos para llevar a cabo esta tarea y además entregar información útil relacionada con proyectos, publicaciones y muestras asociadas a estos genomas. Toda esta información biológica se encuentra almacenada en el NCBI, centro que mantiene información relacionada con secuencias de ácidos nucleicos en la base de datos GenBank, recibiendo datos a través de la colaboración internacional con la *DNA Database of Japan* (DDBJ) y el *European Molecular Biology Laboratory Nucleotide Sequence Database* (EMBL-Bank) además de la comunidad científica, proveyendo sistemas para obtener estos datos y recursos computacionales para el análisis de datos biológicos.

1.1. NCBI y sus bases de datos biológicas

El NCBI es parte de la biblioteca nacional de medicina de los Estados Unidos (NLM) y una rama de los institutos nacionales de salud (NIH), el cual guarda una serie de bases de datos relacionados con biotecnología y biomedicina, además de ofrecer herramientas y servicios para la bioinformática. Dentro de las databases más importantes se encuentra GenBank que almacena secuencias de ADN , PubMed , una base de datos bibliográfica para literatura biomédica y la base de datos Epigenomics que contiene conjuntos de datos epigenéticos[1] para genomas completos[2].

Todas estas bases de datos están disponibles hacia el público mediante la página web de la NCBI que utiliza un motor de búsqueda federado llamado Entrez, un sistema que permite

[1] La epigenética es el estudio de los mecanismos que regulan la expresión de los genes sin una modificación en la secuencia del ADN .

[2] La secuenciación del genoma es uno o varios procesos de laboratorio que determina la secuencia completa de ADN en el genoma de un organismo en un proceso único.

realizar consultas de bases de datos cruzadas, proveyendo acceso a todas las bases de datos de forma simultánea utilizando una interfaz simple de búsqueda, o mediante las utilidades de programación de Entrez que consisten en nueve programas utilizables desde el lado de un servidor para realizar consultas a las bases de datos del NCBI utlizando una URL con sintaxis fija a la cual se le proveen los parámetros necesarios para realizar las búsquedas. Un ejemplo de búsqueda en la web de la NCI puede ser vista en la figura 1.

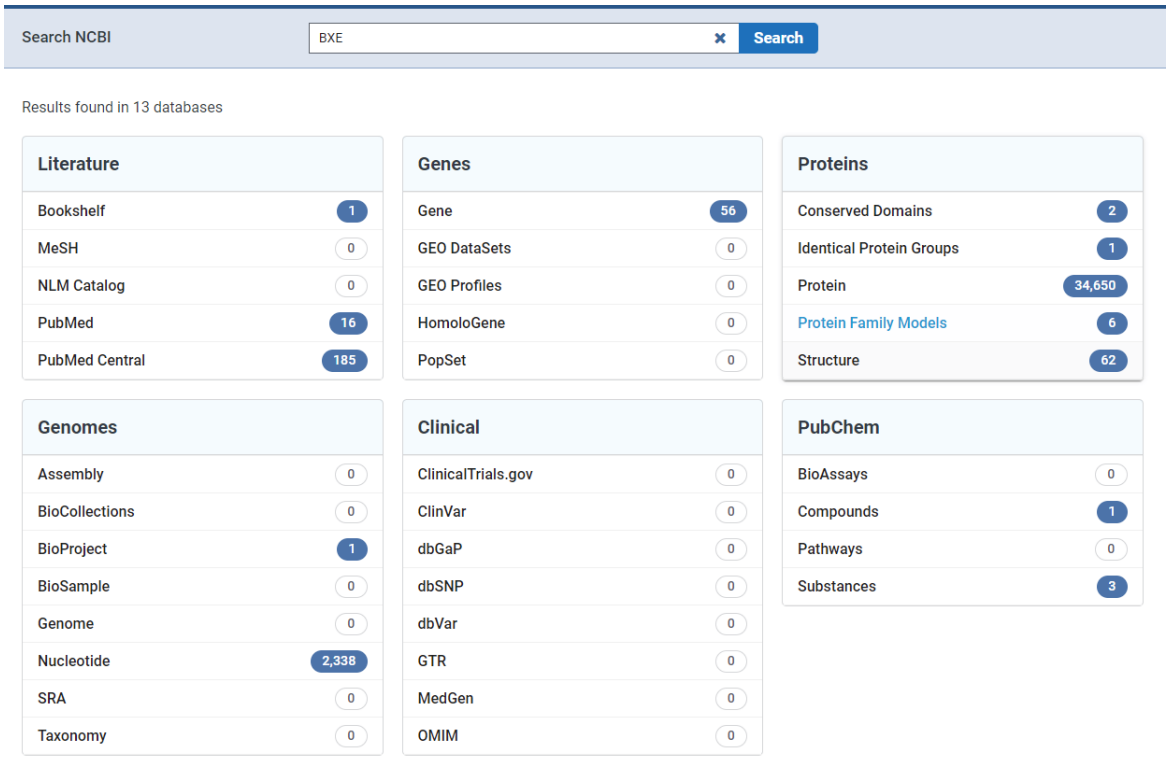

Figura 1: Búsqueda global utilizando el buscador de la página web de la NCBI.
Fuente: NCBI. Search NCBI.

Las cinco bases de datos a integrar corresponden a las bases de datos *Genomes*, BioSample, BioProject, PubMed y *Assembly*, siendo Genomes la base de datos principal de donde se necesitan obtener todos los datos necesarios para graficar los contextos genómicos mientras que el resto contiene información relacionada con publicaciones , nombres y metadatos que complementan estos.

1.1.1. Base de datos *Genomes*

La base de datos Genomes corresponde a un directorio dentro de el servidor FTP, el cual contiene todos los genomas que pueden ser accedidos libremente mediante la página web del NCBI o este servidor. Dentro del directorio cada genoma, que corresponde a un sub-directorio, se encuentran distintos archivos comprimidos o de texto plano con información

relacionada al genoma, como los presentes en la tabla 1, por lo que se debe analizar y delimitar qué información sobre el gen es necesaria para elegir el archivo correcto. Otro problema que trae el tener estos datos en un servidor FTP es que, para obtenerlos, descomprimirlos si es el caso de un archivo comprimido, procesarlos y luego enviarlos hacia el cliente es una tarea que consume mucho tiempo por lo que parte de la solución deberá abarcar el como agilizar este proceso de obtención de la información sobre el genoma y cómo procesarlos para obtener solo lo necesario y evitar entregar información que no es necesaria. Actualmente, el *frontend* encargado de gráficar los contextos genómicos descarga, descomprime y procesa los archivos desde el servidor FTP para poder obtener la información necesaria y luego los archivos completos quedan almacenados localmente, teniendo que procesar nuevamente los mismos si son cosultados nuevamente, y para obtener información adicional desde las otras bases de datos se tiene que ir a la base de datos correspondiente dentro de la NCBI y nuevamente descargar y procesar archivos. Es aquí donde se presenta el problema pincipal y la oportunidad de mejora mediante la integración de los datos y la generación de un esquema que permita obtener todos los datos mediante una única consulta.

Tabla 1: Ejemplo de listado de archivos en el servidor FTP para genoma GCA_000001215. Fuente: Elaboración Propia.

Nombre	Tamaño
GCA_000001215.4_Release_6_plus_ISO1_MT_assembly_report.txt	209K
GCA_000001215.4_Release_6_plus_ISO1_MT_assembly_stats.txt.	15K
GCA_000001215.4_Release_6_plus_ISO1_MT_cds_from_genomic.fna.gz	10M
GCA_000001215.4_Release_6_plus_ISO1_MT_feature_count.txt.gz	610B
GCA_000001215.4_Release_6_plus_ISO1_MT_feature_table.txt.gz	1.5M
GCA_000001215.4_Release_6_plus_ISO1_MT_genomic.fna.gz	42M
GCA_000001215.4_Release_6_plus_ISO1_MT_genomic.gbff.gz	69M
GCA_000001215.4_Release_6_plus_ISO1_MT_genomic.gff.gz	7.5M
GCA_000001215.4_Release_6_plus_ISO1_MT_genomic.gtf.gz	6.8M
GCA_000001215.4_Release_6_plus_ISO1_MT_genomic_gaps.txt.gz	5.6K
GCA_000001215.4_Release_6_plus_ISO1_MT_protein.faa.gz	9.3M
GCA_000001215.4_Release_6_plus_ISO1_MT_protein.gpff.gz	33M
GCA_000001215.4_Release_6_plus_ISO1_MT_rna_from_genomic.fna.gz	16M
GCA_000001215.4_Release_6_plus_ISO1_MT_translated_cds.faa.gz	6.4M
README.txt	43K
annotation_hashes.txt	410B
assembly_status.txt	14B
md5checksums.txt	5.4K

1.2. Trabajo relacionado

El problema de integrar datos heterogéneos se ha buscado resolver de distintas formas, mediante distintas metodologías y acercamientos, buscando constantemente mejorar la calidad de los datos integrados o los tiempos de respuesta para las consultas realizadas.

1.2.1. Integración basada en ontologías

Típicamente una ontología se define como una especificación formal de una conceptualización [Gruber, 1993], un esquema creado para formalizar los datos que son relevantes para el dominio, especificando que es lo relevante y como es expresado de acuerdo con el vocabulario definido en el esquema. Varios autores han propuesto o utilizado sistemas basados en ontologías para realizar la integración de datos heterogéneos, ya sea buscando una mayor precisión con respecto a los datos originales o un menor tiempo de respuesta con respecto a las *Querys* realizadas.

- SOBA: Paul Buitelaar et al. [Buitelaar *et al.*, 2008] presentan el diseño, implementación y evaluación del sistema basado en ontología SOBA, el cual es capaz de procesar información estructurada, texto y pies de foto para extraer información e integrarla en una base de conocimiento coherente, la cual es usada para consultar la información integrada. Además, relaciona la información extraída de las distintas fuentes y detecta la información duplicada e incluso utiliza procesamiento de lenguaje natural para incrementar la robustez y precisión, llegando a obtener precisión del 88 % para las instancias donde se tiene una gran cantidad de información.

- RDF basado en ontología: Maxime Buron et al. [Buron *et al.*, 2019] introducen una novedosa clase de RDF llamado RDF Integration Systems (RIS), permitiendo exponer, integrar y flexibilizar las consultas a los datos que provienen de las fuentes de datos heterogéneas mediante GLAV , una opción que generaliza y maximiza la flexibilidad de las opciones existentes para relacionar los esquemas locales con el esquema global como lo son GAV que define cada relación como una vista sobre los esquemas locales y LAV donde los elementos de los esquemas locales son definidos como vistas sobre el esquema global. Esta implementación ha obtenido tiempos de respuesta con respecto a las consultas que van desde los 10 milisegundos hasta los 105 milisegundos , dependiendo de la complejidad de la consulta y la metodología utilizada.

- Chaimaa Messaoudi et al.[Messaoudi *et al.*, 2020] proponen un sistema con semántica basada en ontología para trabajar con datos biológicos relacionados con las proteínas, todo esto utilizando un acercamiento como mediador y el framework de Apache Spark para realizar las transformaciones a las *querys* y cuestionar las fuentes de datos. Este acercamiento logró tiempos de respuesta entre 195 segundos hasta 3673 segundos para las consultas de mayor tamaño.

1.2.2. Otros Acercamientos

- Venkatar Ramesh et al.[Ramesh y Ram, 1997] propusieron una metodología de integración de datos utilizando restricciones de integridad, las cuales son aplicables a las bases de datos locales. Además introduce el concepto de relaciones basadas en restricciones entre objetos provenientes de bases de datos heterogéneas y como las restricciones generadas facilitan el procesamiento semántico de las consultas.

- Vadim Y. Bichutskiy et al.[Bichutskiy *et al.*, 2006] proponen una estrategia híbrida para integrar datos heterogéneos provenientes de bases de datos de investigación del cáncer P53. Este acercamiento busca combinar la utilización de un mediador, el cual acepta una consulta del cliente, determina las fuentes a las que debe acceder y luego descompone la consulta en subconsultas para cada fuente requerida, y un almacén de datos (Data Warehousing) centralizado donde se diseña un esquema global que combina los esquemas de las fuentes de datos, todo esto generando una base de datos que es mediador y almacén de datos a la vez.

- Byoung-Ha Yoon et al.[Yoon *et al.*, 2017] proponen el uso de una base de datos tipo Grafo para la integración de datos biológicos heterogéneos utilizando Neo4j, recolectando datos de distintas fuentes y removiendo los datos redundantes y duplicados para construir la base de datos para luego comparar el tiempo de respuesta con una base de datos MySQL. Con esta implementación se lograron tiempos de respuesta de casi 25 segundos para las consultas más complejas y de casi 0 segundos para las consultas un poco más simples, tiempos mucho menores que los registrados por la base de datos MySQL.

- H. Ulrich et al. [Ulrich *et al.*, 2019] buscaron definir una interfaz uniforme que pueda comunicarse con los distintos repositorios de metadatos existentes y por lo tanto crear un modelo de datos estandarizado. Para esto se utilizó el lenguaje de consultas tipo GraphQL QL4MDR, compatible con la ISO 11179-3, un estándar internacional para la representación de metadata, con la cual se guiaron para realizar la integración de los datos presentes en repositorios de datos que cumplen parcialmente con las normas como lo es esta ISO.

- Patrick Stünkela et al.[Stünkel *et al.*, 2020] proponen un acercamiento basado en modelo, utilizando un modelamiento multi vista y un modelamiento del dominio específico, siendo este modelo implementado y presentado mediante el framework GraphQL. En esta propuesta se busca relacionar los distintos esquemas existentes mediante referencias cruzadas que no modifican ninguno de estos esquemas, las cuales son identificadas mediante llaves, para luego juntar los esquemas y generar uno global. Para consultar información de este nuevo esquema se construyeron resolvers, los cuales utilizan los resolvers de los endpoints ya existentes para construir la data integrada, pero buscando reducir la cantidad de consultas realizadas, esto utilizando las llaves creadas anteriormente para combinar recursivamente los objetos consultados si es que poseen la misma llave.

1.3. Objetivos

1.3.1. Objetivo General

Diseñar e implementar una API mediante el lenguaje de consultas GraphQL utilizando esta aplicación como un mediador para abstraer las bases de datos de la NCBI accedidas e integrar los distintos datos biológicos que se encuentran en estas.

1.3.2. Objetivos Específicos

- Diseñar el esquema que integrará los datos.
- Diseñar un mecanismo para abstraer y mapear los datos heterogéneos.
- Implementar una API utilizando el esquema diseñado.
- Evaluar el rendimiento de la solución implementada.

CAPÍTULO 2
MARCO CONCEPTUAL

2.1. API

La WWW (*World Wide Web*) expone datos en una de las dos posibles formas existentes. La primera forma es exponer data mediante documentos HTML que se encuentran almacenados en un servidor web. La segunda manera de exponer data es a través de interfaces máquina-a-máquina. *Application Programming Interface* o API, corresponde a una interfaz que puede ser utilizada por un programa de software que interactúa con una aplicación existente, en este caso un software de visualización, la cual permite acceder a datos de una aplicación existente, siendo las dos bases de datos heterogéneos las fuentes de datos a las que se quiere acceder. Estas interfaces exponen las funcionalidades y datos de una aplicación a través de internet, enviando datos de ida y vuelta mediante peticiones HTTP, las cuales generalmente retornan datos en formato XML o JSON, permitiendo que dos entidades se comuniquen la una con la otra en un formato más estandarizado. La figura 2 muestra una estructura básica de una API.

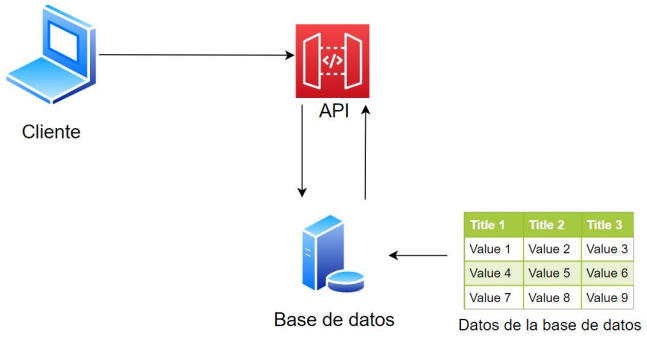

Figura 2: Ejemplo de una estructura básica de una API.
Fuente: Elaboración propia.

2.1.1. Servicios webs

Los servicios webs son un subconjunto de las APIs, por lo que todo servicio web es una API, pero no toda API es un servicio web. Esto se debe a que una API puede ser implementada *online* u *offline*, y puede usar cualquier tipo de protocolo o estilo de diseño, mientras que un servicio web, por definición, requieren de una red, lo que implica que necesitan seguir una gran cantidad de estándares. Estos servicios web utilizan usualmente protocolos como SOAP y estilos de diseño como REST, además de ser asociados con arquitecturas orientadas a servicios.

2.1.2. REST

Representational State Transfter [Fielding y Taylor, 2000] o REST, es una de las interfaces más prevalentes y utilizadas por los servicios webs, la cual es construida sobre el estándar HTTP [Fielding *et al.*, 1999] lo que implica ciertas restricciones provenientes de este tipo de arquitectura. Estas interfaces buscan proveer las siguientes características:

- Un modelo cliente-servidor: la interfaz del usuario está separada de los datos que ocupa el servicio, eso para promover la separación de responsabilidades entre el cliente y el servidor, en cambio una API solo provee funcionalidad.
- Cacheabilidad [3] : una característica que permite marcar datos como cacheables, lo deja al caché del cliente determinar si una *request* se debe realizar nuevamente o si los datos almacenados localmente por el cliente son suficientes.
- *Stateless*[4]: Las *request* necesitan incluir todos los datos necesarios para entenderla, de esta forma las *requests* que son idénticas obtienen un resultado idéntico. Esto promueve la confiabilidad para recuperarse de fallas, visibilidad para los registros y escalabilidad con respecto al manejo de recursos, haciendo que el servicio web no necesite manejar estos para lograr responder a N clientes.
- Una interfaz uniforme: que es definida por cuatro restricciones de interfaz, las cuales provienen de la arquitectura de una REST API. Estas corresponden a la manipulación de recursos mediante representaciones, mensajes auto-descriptivos, identificación de recursos y el uso de *hypermedia*[5] como motor del estado de una aplicación. Estas restricciones significan que los datos son descritos como recursos, identificadores de recursos, representaciones, meta datos, representación de meta datos y datos de control.
- Arquitectura por capas: La arquitectura REST debe restringir el conocimiento de componentes dentro de la arquitectura de tal forma que un componente no puede ver o interactuar con servicios con los que no está conectado inmediatamente.

Además se tiene una restricción opcional, la cual indica que un servidor puede expandir la funcionabilidad de los clientes proporcionando código en forma de *applets*[6] o *scripts*[7].

Todas las restricciones descritas anteriormente son parte del estilo arquitectónico de REST y por lo tanto no son especificaciones, ya que la implementación de una API con arquitectura REST puede sufrir variaciones en los requerimientos y por lo tanto no cumplen con todas

[3]Refiriéndose a la capacidad de almacenar datos a una alta velocidad, pero que no son duraderos.
[4]Que no posee un estado.
[5]Es una extensión del término hipertexto, permitiendo incluir gráficos, audio, texto plano y hyperlinks.
[6]Programa que puede incrustarse en una página Web
[7]Lista de comandos que son ejecutados por un cierto programa o motor de scripts

las restricciones, en cambio una especificación describe exactamente y detalladamente la naturaleza de la representación de los datos, sus protocolos de comunicación y mensajería, como lo es el protocolo SOAP [Box *et al.*, 2000].

2.1.3. *Service Oriented Architectures*

Service Oriented Architectures o SOA corresponde a un patrón arquitectónico para diseñar aplicaciones de software donde las características estas separadas y son puestos a disposición como servicios dentro de una red. Los objetivos de un SOA son similares a los de una arquitectura REST, los cuales son:

- Una SOA necesita ser dinámicamente descubrible en tiempo de ejecución y basado en los criterios de la aplicación.
- Interoperabilidad: Una SOA necesita ser capaz de comunicarse usando una plataforma y un lenguaje de programación de manera independiente.
- Auto-contenido y modular: Una SOA necesita que los servicios ofrecidos puedan ser descompuestos en conjuntos de funcionalidades, los cuales pueden combinar para proveer de interacciones más complejas. Estos servicios ocultan los detalles de su implementación y están aislados de posibles interrupciones que puedan suceder en los otros servicios con los que está compuesto.
- Esta arquitectura debe tener muy pocas dependencias conocidas entre el consumidor de la SOA y su proveedor.
- Una SOA debe ser capaz de recuperarse de errores sin la intervención humana.
- Las SOAs deben proveer métodos de interacción idempotententes, es decir, que siempre produzcan el mismo resultado si los métodos son llamados de la misma manera.

Estos tipos de arquitectura son generalmente descritos por una especificación conocida como *Web Service Description Language* o WSDL [Christensen *et al.*, 2001], el cual es un lenguaje que provee una representación sintáctica para describir las capacidades y entidades lógicas sobre las cual trabaja el servicio web, esto quiere decir que entrega la información suficiente para que un servicio consumidor pueda entender los servicios, parámetros y la estructura de los datos que son retornados. Ya que WSDL solo entrega una descripción estructural del servicio web se necesita de una descripción semántica para los datos disponibles, es aquí donde aparece el concepto llamado web semántica [Berners-Lee *et al.*, 2001]. La idea es describir como los datos existen y operan dentro del contexto de un servicio web, describiendo clases de objetos y las relaciones entre ellos junto con un conjunto de reglas de inferencia. Dentro de los principales intentos para crear una web semántica se encuentran RDF [Klyne y Carroll,] y OWL [Bechhofer *et al.*, 2009].

2.2. GraphQL

GraphQL es un lenguaje de consultas para las APIs que se ejecuta desde el lado del servidor para poder realizar consultas utilizando un esquema definido por el desarrollador de la API. Las consultas a este esquema que se define son realizadas en el lenguaje de GraphQL, utilizando una estructura parecida a un JSON el cual describe atributos y relaciones. Un ejemplo de estas consultas puede ser vista en la figura 3.

```
query {
    genome {
        _id
    }
    features{
        location
    }
}


{
    "data": {
        "genome" {
            "_id": "NT_037436"
        }
        "features": [
            {"location": "1..2" },
            {"location": "2..4" },
            ..
        ]

    }
}
```

Figura 3: Ejemplo de una *Query* y su resultado en el lenguaje GraphQL.
Fuente: Elaboración propia.

El esquema que se construye con GraphQL representa todos lo datos en formato JSON, por lo que todas las consultas que provengan desde un servicio web deben ser transformados adecuadamente para realizar la consulta y luego retornar un JSON. En la figura 4 se tiene un ejemplo de un esquema de GraphQL en donde se definen los distintos tipos con sus atributos, además de una interfaz que permite compartir atributos entre los tipos una enumeración de ítems, una unión de tipos y una Query como la vista anteriormente.

```
type Genome {
    _id: String!
    definition: String!
    features: [Feature]!
}

type Feature {
    _id: ObjectId!
    location: String!
    locus_tag: String
}

union SearchResult = Genome | Feature

type Query {
    getGenome(id: String!): Genome
    feature(id: ObjectId!): Feature
}
```

Figura 4: Ejemplo de un esquema en GraphQL.
Fuente: Elaboración propia.

Al tener un esquema con un tipeado fuerte se puede determinar de manera más fácil si ciertos datos están disponibles y de qué forma existen, lo que hace la API menos propensa a errores. Además GraphQL nos permite realizar consultas más rápidas y que consumen menos recursos de red, ya que permite consultar solo lo necesario de los elementos pedidos y en una única Query , al contrario de las arquitecturas REST las cuales realizan una Query por cada elemento que se pide además de entregar todos sus atributos, algo que afecta el rendimiento y aumenta el consumo de recursos, algo que es caro para los usuarios.

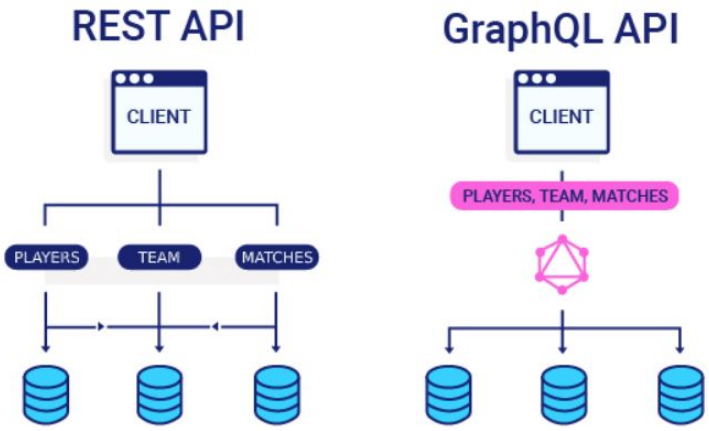

Figura 5: Arquitectura simple de una API REST y una API de GraphQL.
Fuente: GraphQL vs REST. Devathon.

Como se puede ver en la figura 5, la arquitectura de una API REST y una API de GraphQL son muy similares a grandes rasgos, pero se ve una gran e inmediata diferencia en la forma de pedir y entregar datos, mientras la API REST necesita comunicarse con las distintas fuentes de datos mediante las distintas *requests*, GraphQL permite que el cliente realize una única *request*, lo que puede ser visto a grandes rasgos como una integración de datos, que es el objetivo de este trabajo.

2.2.1. Manejo de recursos

GraphQL tiene el concepto de recursos al igual que la arquitectura REST, identificando estas mediante URLs, estableciendo una comunicación donde se recuperan los recursos mediante solicitudes y se obtienen datos en formato JSON como respuesta. En GraphQL, a diferencia de REST, la forma en que se obtiene un recurso esta separada de la identidad del mismo, ya se define *types* para la información que se quiere obtener y un esquema que especifica esta información, siendo el tamaño de ésta definida por la *query* realizada.

2.2.2. Esquema

El esquema que utiliza GraphQL, como el visto en la figura 4, define dos tipos iniciales como lo son *Query* utilizado para obtener datos y *Mutations* utilizadas para modificar datos.A simple vista estos tipos son similares a las consultas de una arquitectura REST, pero con GraphQL y su esquema se pueden generar consultas más complejas que incluyen datos adicionales o consultas anidadas. Este esquema utiliza SDL, con el cual define los tipos dentro de la API. Esto permite que después de definir un esquema el equipo de *frontend* y *backend* puede trabajar en paralelo, ya que se conoce exactamente la estructura de los datos, aumentando la productividad del equipo.

2.2.3. *Resolvers*

Las funciones que utiliza GraphQL para manejar las llamadas a la API son llamadas *resolvers* y al igual que en un arquitectura REST y sus *route handlers* busca ejecutar cierto código y funciones para producir un resultado que será devuelto a quien realizó la llamada. La diferencia es que GraphQL no implementa funciones para URLs específicas sino que para un campo específico dentro de un tipo como lo es una Esto permite que las llamadas a la API puedan procesar múltiples campos o un mismo campo múltiples veces en una única *request*.

Figura 6: Ejemplo de una multiples llamadas en una arquitectura REST.
Fuente: GraphQL vs REST. Devathon.

2.2.4. Obtención de datos

Como fue visto anteriormente, las arquitecturas REST obtienen los datos mediante múltiples llamadas a la API para obtener específicamente lo que necesitan, como se puede ver en la figura 6, mientras que GraphQL obtiene la misma cantidad de información mediante una única llamada.

GraphQL resuelve el problema de *over-fetching*, el cual consiste en obtener datos que no son necesarios para la aplicación que realiza una llamada a la API, y el problema de *under-fetching*, que consiste en no obtener todos los datos necesarios en una llamada a la API, por lo que son necesarias más de una para obtener todos los datos que se requieren. Ambos problemas están presentes en la arquitectura REST, pero gracias a la forma en que se realizan consultas mediante el esquema de GraphQL se puede obtener exactamente la información que se consulta, un ejemplo de esto se puede ver en la figura 7.

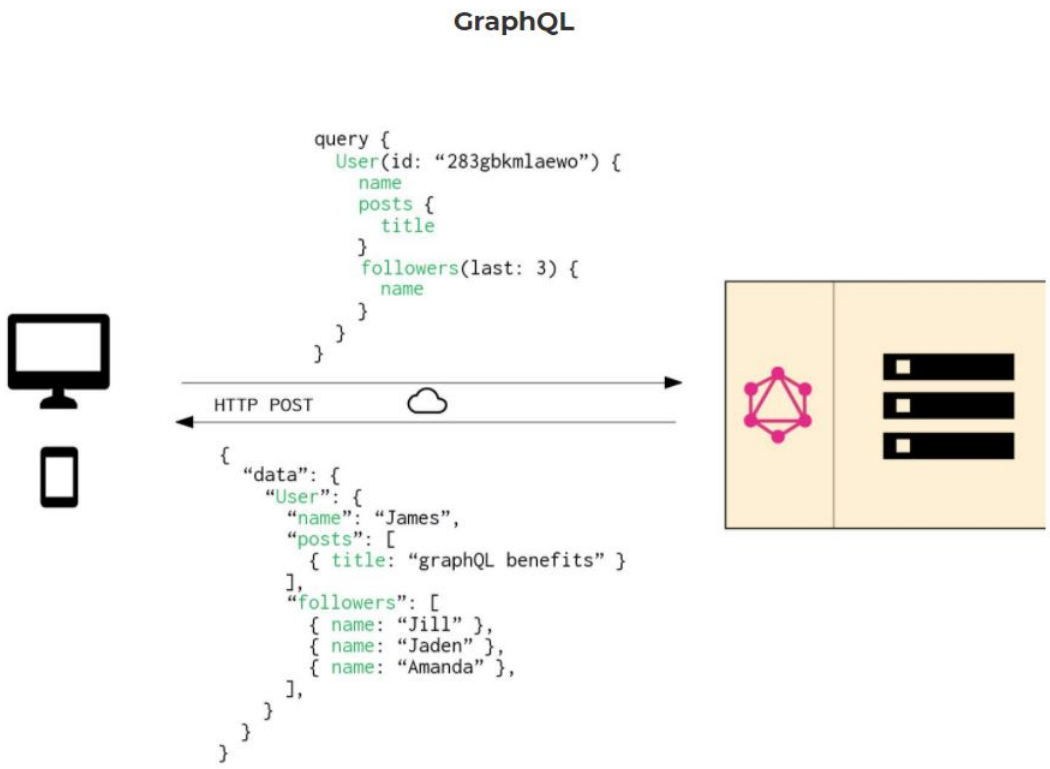

Figura 7: Ejemplo de una llamada utilizando el esquema de GraphQL.
Fuente: GraphQL vs REST. Devathon.

2.2.5. Manejo de errores y permisos

En las arquitecturas REST, al trabajar sobre el protocolo HTTP, puede hacer uso de los códigos de estado de HTTP como lo son el código 200 que significa que esta todo *OK* o el código 404 que significa *Not Found*. Esto permite manejar los errores y permisos entregando como respuesta alguno de estos códigos cuando sea necesario, pero esto también implica que si se piden varios recursos dentro de los cuales solo se tiene permiso a algunos, la respuesta final que se entregará al cliente es la de un error de permisos junto con una respuesta vacía de datos. Es aquí donde GraphQL hace las cosas diferentes, ya que al no utilizar necesariamente una protocolo estandarizado para realizar llamadas a la API, se pueden obtener datos parciales a los que se tiene permiso de acceso, mientras que los recursos a los que no se puede acceder simplemente son omitidos y se agrega una arreglo de errores que indica exactamente el campo al que no se tiene permiso, como se puede ver en la figura 8. El problema de esto es que, como fue explicado anteriormente, el formato de los errores no sigue ningún estándar, por lo que el error entregado puede ser cualquier cosa que el desarrollador quiere que sea, implicando que se debe tener un mayor cuidado al definir estos para no tener problemas con la comunicación entre el *frontend* y el *backend*.

```
{
    "data": {
        "users" : [
            {
                "name": "Camilo P",
                "address": "User address",
            },
            {
                "name": "Carlos P",
                "address": null,
            },
            ...
        ]
    },
    "errors": [
        {
            "message": "permision denied for Field address",
            "locations": [
                {
                    "line": x,
                    "column": y,
                },
                ...
            ]
        },
    ]
}
```

Figura 8: Ejemplo de una respuesta con datos parciales en GraphQL.
Fuente: Elaboración propia.

2.3. NCBI

Las bases de datos genéticas con las que se trabajará forman parte de la NCBI y son utilizadas para guardar datos como genes, productos de genes, variantes, fenotipos, etc, los cuales son almacenados y permiten a un usuario recuperar y agregar datos además de extraer información de estos. Son repositorios de datos organizados los cuales funcionan como una fuente de datos para entender el funcionamiento de los organismos. Además de datos con información genética específica de un organismo también se almacenan publicaciones, revistas, artículos y se proveen de herramientas para trabajar con estos como BLAST, una herramienta que nos permite encontrar regiones de similitud entre secuencias biológicas, Entrez, el motor de búsqueda que utiliza la NCBI para realizar búsquedas avanzadas a lo largo de to-

das la bases de datos, el servidor FTP para acceder directamente a las bases de datos, entre otras[8]. En este trabajo nos enfocaremos en las bases de datos pedidas para integrar y en la herramienta de Entrez que nos permite navegar entre estas.

2.3.1. Entrez

Entrez es un sistema de bases de datos que proporciona acceso integrado a una gran variedad de recursos existentes dentro de la NCBI incluyendo las bases de datos de *Genome*, *BioSample*, *BioProject*, *Assembly*, *Pubmed y Nucleotide*, es un motor de búsqueda federado[9] que permite realizar búsquedas utilizando su interfaz de usuario intuitiva mediante el uso de *strings* que sirven como filtro de estas y cuyo resultado puede ser visualizado en distintos formatos como FlatFile, Fasta, XML , entre otros. Este sistema además nos proporciona la herramienta de *Entrez Programming Utilities* o Eutils, que nos permite tener un acceso más directo a los resultados de las consultas utilizando una API con distintas funcionalidades que están definidas mediante URLs con formato fijo. Existen nueve *E-Utilities* dentro de entres:

- EInfo: Proporciona el número de registros indexados en cada campo de una base de datos determinada, la fecha de la última actualización de la base de datos y los enlaces disponibles desde la base de datos a otras bases de datos Entrez. Utiliza la URL fija `eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi`.
- ESearch: Responde a una consulta de texto con la lista de UIDs coincidentes en una base de datos determinada para su uso posterior en ESummary, EFetch o ELink. Utiliza la URL fija `eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi`.
- EPost: Acepta una lista de UIDs de una base de datos determinada, almacena el conjunto en el Servidor de Historias y responde con una clave de consulta y un entorno web para el conjunto de datos cargado. Utiliza la URL fija `eutils.ncbi.nlm.nih.gov/entrez/eutils/epost.fcgi`.
- ESummary: Responde a una lista de UIDs de una base de datos dada con los correspondientes resúmenes de documentos. Utiliza la URL fija `eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi`.
- EFetch: Responde a una lista de UIDs en una base de datos dada con los registros de datos correspondientes en un formato especificado. Utiliza laURL fija `eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi`.
- ELink: Responde a una lista de UIDs en una base de datos dada con una lista de UIDs relacionados en la misma base de datos o una lista

[8]Una lista completa de los recursos entregados por la NCBI puede encontrarse en `https://www.ncbi.nlm.nih.gov/guide/all/`.

[9]Recupera información de diversas fuentes a través de una aplicación de búsqueda construida sobre uno o varios motores de búsqueda.

de UIDs vinculados en otra base de datos de Entrez. Utiliza la URL fija `eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi`.

- EGQuery: Responde a una consulta de texto con el número de registros que coinciden con la consulta en cada base de datos Entrez. Utiliza la URL fija `eutils.ncbi.nlm.nih.gov/entrez/eutils/egquery.fcgi`.
- ESpell: Recupera sugerencias ortográficas para una consulta de texto en una base de datos determinada. Utiliza la URL fija `eutils.ncbi.nlm.nih.gov/entrez/eutils/espell.fcgi`
- ECitMatch: Recupera los ID de PubMed correspondientes a un conjunto de cadenas de citas de entrada. Utiliza la URL fija `eutils.ncbi.nlm.nih.gov/entrez/eutils/ecitmatch.cgi`.

Cada una de estas funcionalidades tiene parámetros únicos como también parámetros que comparten algunas de estas, como lo es **db**, el parámetro que indica la base de datos objetivo, un parámetro que siempre debe estar presente en las consultas y cuyo valor es una de las bases de datos que maneja Entrez. Una lista de las bases de datos que maneja Entrez con una pequeña descripción puede verse en el anexo 2.

2.3.2. *Assembly*

La base de datos *Assembly* contiene información acerca de la estructura de los genomas ensamblados[10], los que se pueden presentar en un archivo AGP[11] o en una colección de cromosomas secuenciados. Además proporciona un *assembly accession number* versionado, un identificador único que rastrea los cambios en los *assemblies* a medida que son actualizados a lo largo del tiempo. Esta base de datos rastrea la relación entre un *assembly* registrado en la International Nucleotide Sequence Database Collaboration (INSDC) y un *assembly* representado en el proyecto *NCBI Reference Sequence* o RefSeq.

La base de datos representa genomas ensamblados a diferentes niveles:

- Ensamblajes completos del genoma.
- Ensamblajes que incluyen cromosomas o grupos de enlace, *scaffolds*[12] y *contigs*[13].
- Ensamblajes que incluyen *scaffolds* y *contig*.

[10] Se refiere al alineamiento y mezcla de múltiples fragmentos de una secuencia de ADN mucho mayor para reconstruir la secuencia original.

[11] Archivo que describe el ensamblaje de un objeto de secuencia mayor a partir de objetos más pequeños.

[12] Una serie de contigs, que están en el orden correcto pero no necesariamente conectados en un tramo continuo de secuencia.

[13] Segmentos de ADN superpuestos, que juntos representan una región consenso de ADN.

- Ensamblajes que incluyen sólo *contigs*.

Y estos *assembly* pueden ser buscados utilizando campos como:

- Un nombre de organismo o especie.
- Un *assembly accession*.
- Un nombre de ensamblaje, o un sinónimo como por ejemplo, hg19, GRCh37, GRCh38.p4

2.3.3. *Bioproject*

La base de datos *BioProject* provee una manera de acceder a información sobre proyectos con referencias hacia datos que se han depositado o se depositarán en bases de datos de archivos mantenidas por los miembros del International Nucleotide Sequence Database Consortium (INSDC), que comprende la DDBJ, el Archivo Europeo de Nucleótidos en el *European Molecular Biology Laboratory* (ENA), y el GenBank en la NCBI. Los proyectos almacenados son creados por iniciativas que generan un volumen muy grande de datos , datos provenientes de múltiples miembros de un consorcio o colaboración, o datos que se envían a múltiples bases de datos. Estos describen esfuerzos de investigación a gran escala y pueden ser enviados a la NCBI directamente o a otras bases de datos asociadas con el INSDC y luego citando el identificador del proyecto, su *accession*, lo que hace a la base de datos *BioProject* una forma robusta de acceder a datos que se encuentran a lo largo de múltiples recursos y múltiples tiempos de entrega.

La definición de un proyecto es muy flexible, por lo que se pueden crear una variedad de proyectos complejos y varios sub proyectos dentro. Estos pueden ser establecidos para:

- Secuenciación y ensamblaje del genoma.
- Metagenomas[14].
- Secuenciación y expresión del transcriptoma[15].
- Secuenciación de locus dirigidos.
- Mapas genéticos.
- Epigenómica y genómica funcional.

[14]Estudio del material genético, el cual es obtenido directamente de muestras ambientales.

[15]Conjunto de todas las moléculas de ARN presentes en una célula o grupo de células en un momento determinado.

- Fenotipo[16] o genotipo[17].
- Detección de variaciones.

2.3.4. *BioSample*

La base de datos de *BioSample* almacena información descriptiva acerca de los materiales biológicos que están presentes en los datos principales que almacena la NCBI, albergando diversos tipos de muestras de cualquier especie. Esta base de datos promueve el uso de nombres y valores de atributos estructurados y coherentes que describen lo que son las muestras, así como información sobre su procedencia. Esta información es importante para proporcionar un contexto a los datos, para que así puedan comprenderse mejor. Además promueve la reutilización y permite la agregación e integración de conjuntos de datos dispares, facilitando nuevos conocimientos y descubrimientos en una amplia gama de campos biológicos.

Los registros de esta base de datos y al igual que las demás usadas en este trabajo, están indexados, por lo que permiten realizar búsquedas y a la vez estas incluyen datos que referencian a otras bases de datos, reduciendo la carga del usuario que hizo la consulta al no contener toda esta información en la muestra sino que en forma de referencia. Las bases de datos que están vinculadas a la de *BioSample* son:

- *BioProject*.
- *Sequence Read Archive* (SRA).
- *Gene Expression Omnibus* (GEO).
- *database of Genotypes and Phentotypes* (dbGaP).

además de secciones de la base de datos *GenBank* como:

- *Expressed Sequence Tags* (EST).
- *Genome Survey Sequences* (GSS).
- *Whole Genome Shotgun*[18] (WGS).
- *Transcriptome Shotgun Assembly* (TSA).

[16] Cualquier característica o rasgo observable de un organismo.
[17] Información genética que posee un organismo en particular, en forma de ADN.
[18] *Shotgun sequencing* es un método utilizado para secuenciar cadenas de ADN aleatorias.

2.3.5. PubMed

PubMed es un recurso dentro de la NCBI que apoya la búsqueda y recuperación de literatura biomédica y de ciencias de la vida con el objetivo de mejorar la salud, tanto a nivel global como personal. Esta base de datos contiene una gran cantidad de citaciones que provienen principalmente de los campos de la biomedicina y la salud además de disciplinas relacionadas como las ciencias de la vida, las ciencias del comportamiento, las ciencias químicas y la bioingeniería.

Los recursos literarios a los que PubMed nos permite acceder dentro de la NLM son:

- MEDLINE: el mayor componente de PubMed y consiste principalmente en citas de revistas seleccionadas para MEDLINE; artículos indexados con MeSH y curados con metadatos de financiación, genéticos, químicos y otros.
- PubMed Central (PMC): archivo de texto completo que incluye artículos de revistas revisadas y seleccionadas por la NLM para su archivo, así como artículos individuales recogidos para su archivo en cumplimiento de las políticas de los financiadores.
- Bookshelf: archivo de textos completos de libros, informes, bases de datos y otros documentos relacionados con la biomedicina, la salud y las ciencias de la vida.
- Referencias antiguas de la versión impresa del *Index Medicus*[19], desde 1951 y antes.
- Referencias a algunas revistas antes de que fueran indexadas en el *Index Medicus* y MEDLINE, por ejemplo *Science*, BMJ y *Annals of Surgery*.

2.3.6. *Nucleotide*

La base de datos de nucleótidos es simplemente una colección de secuencias procedentes de varias fuentes, como GenBank, RefSeq, TPA y PDB. Los datos de las secuencias del genoma, los genes y las transcripciones constituyen la base de la investigación y el descubrimiento biomédicos.

Los datos que se quieren integrar provienen de todas las bases de datos anteriormente descritas, además de la base de datos *Genomes*,los cuales serán consumidos por un software de visualización que busca visualizar genomas bacterianos e identificar contextos genómicos en los mismos. Para la identificación de contextos genómicos se busca analizar agrupaciones génicas conocidas como *gene clusters* los cuales son necesarios para entender qué genes se encuentran en las vecindades de una secuencia codificante (CDS) de interés, si esta organización se encuentra conservada en otros organismos (sintenia[20]) y si tienen relevancia a las

[19]Subconjunto curado de MEDLINE.

[20]La sintenía describe la co-localización física de un marcador genético en el mismo cromosoma dentro de un individuo o especie.

funciones asociadas a la CDS de estudio. Por otro lado, tenemos la visualización de genomas bacterianos, los cuales se representan mediante gráficos circulares. los cuales muestran de forma global algunos parámetros genómicos, información que se obtiene de los datos que se obtienen de las bases de datos presentadas anteriormente. Un ejemplo de esta visualización se puede ver en la figura 9.

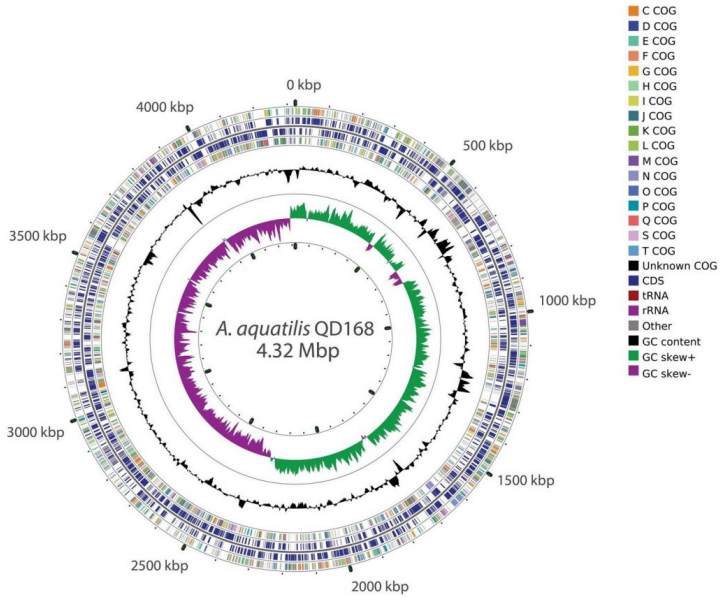

Figura 9: Representación visual de un genoma bacteriano realizada mediante CGview y editado manualmente mediante Adobe illustrator.
Fuente: Aplicación de aprendizaje de máquinas y minería de texto para el análisis de datos genómicos y desarrollo de software bioinformático (PI_M_2020_43). UTFSM.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

3.1. Metodología de trabajo

Ya que se trata de un proyecto individual, la metodología de trabajo utilizada consistirá en realizar las siguientes etapas, las cuales nos permitirán tener un orden y objetivos que cumplir durante el desarrollo del proyecto:

- Toma, documentación y verificación de requisitos: etapa que consiste en entender los requisitos que presenta el *frontend* encargado de graficar los contextos genómicos, la información que necesita de cada base de datos, consultas y posibles restricciones.
- Análisis y Diseño: Análisis de los requisitos, fuentes de datos, herramientas, diseño de modelo para la extracción de datos desde la NCBI y esquema para la integración de las bases de datos, todo esto cumpliendo con los requisitos y objetivos planteados.
- Implementación de la solución: Implementar el esquema diseñado en la etapa anterior.
- Pruebas: realización de casos de prueba para determinar el rendimiento de la solución implementada y que esté funcionando correctamente en todos los casos posibles de consultas realizadas por el *frontend*.

3.2. Toma, documentación y verificación de requisitos

3.2.1. Datos

El *frontend* encargado de graficar los contextos genómicos requiere principalmente información de la base de datos *Genome*, que contiene una descripción completa de cada genoma junto a todas las características que lo componen. Desde este archivo lo que se pide obtener son los siguientes datos:

- *DEFINITION*: corresponde a la definición de la secuencia del genoma, una pequeña descripción que es parte obligatoria del archivo.
- *ACCESSION*: identificador único de la secuencia del genoma, el cual es recomendado de usar para cuando se necesita citar información de GenBank, y al igual que *DEFINITION* es una sección obligatoria del archivo.

- *FEATURES*: tabla que contiene información sobre porciones de la secuencia que codifican proteínas y moléculas de ARN e información sobre sitios de importancia biológica determinados experimentalmente.
- *Feature key*: una sola palabra o abreviatura que indica el grupo funcional.
- *location*: instrucciones para encontrar la característica, puede contener operadores o descriptores funcionales que especifican lo que debe hacerse a la secuencia para reproducir la característica.
- *mobile_element_type*: tipo y nombre o identificador del elemento móvil que es descrito por la característica principal.
- *locus_tag*: un identificador sistemático y estable, suministrado por el remitente, para un gen y sus características asociadas, utilizado con fines de seguimiento.
- *gene*: símbolo del gen correspondiente a una región de la secuencia.
- *product*: nombre del producto asociado a la característica.
- *translation*: secuencia de aminoácidos abreviada de una letra generada automáticamente, derivada del código genético universal o de la tabla especificada en el calificador */transl_table*.

Una descripcion completa de los *Feature qualifiers* puede ser encontrada en `https://www.insdc.org/documents/feature-table`.

Las otras bases de datos correspondientes a *PubMed*, *Assembly*, *BioProject* y *BioSample*, contienen información de publicaciones, muestras, contribuciones, datos relacionados con sus autores y con datos del genoma como a qué especie corresponde o su ubicación en el servidor FTP. Desde estas bases de datos se pide obtener información que sea relevante para el genoma y publicaciones asociadas para que la búsqueda de estas esté integrada en la misma consulta y no se requiera de la realización de más búsquedas dentro del NCBI para encontrarlas.

3.2.2. Consultas

Las consultas que el *frontend* debe realizar al sistema con los datos integrados están definidas de la siguiente manera:

- Una consulta que incluya como parámetro el *Assembly Accession Number*,[21] un identificador para una colección de registros de genomas, además de un *locus tag* de inicio

[21] Assembly accession numbers son números de acceso a la secuencia con un formato distinto que el personal del NCBI asigna a los ensamblajes genómicos individuales. A diferencia de otros números de accesión de secuencias, los *accessions* de ensamblaje no representan un único registro de secuencias, sino la colección de registros de secuencias que componen un ensamblaje genómico individual.

y un *locus tag* de final, los cuales corresponden al identificador único de un gen y que se encuentran como una propiedad opcional dentro de los *features*, en el archivo de texto plano de un genoma. Esta consulta debe retornar como resultado el genoma con la información pedida anteriormente junto con todos los features que se encuentren entre el *locus tag* de inicio y el *locus tag* de final, junto con la información relacionada a este genoma que se encuentre en las bases de datos de *PubMed*, *Assembly*, *BioProject y BioSample*.

- La segunda y último tipo de consulta pedida es una que, dado un *locus tag*, un *Assembly Accession Number* opcional, un número que indique el límite superior y otro que indique el límite inferior, se entregue el primer genoma que tenga este *locus tag* junto con los x *features* anteriores, siendo x el parámetro correspondiente al límite inferior y los y *features* siguientes, siendo y el parámetro correspondiente al límite superior, y al igual que con la primera consulta, información obtenida de las otras bases de datos, relacionados con el genoma obtenido.

3.3. Análisis y Diseño

Ahora que ya se tienen los datos y tipos de consultas requeridas, procedemos a analizar la información que está disponible en la NCBI, el cómo se obtienen actualmente para el uso del *frontend*, cómo se puede mejorar, diseñar el esquema de GraphQL que nos permita integrar los datos y qué tecnologías usar para lograrlo.

3.3.1. Análisis y extracción de datos genómicos

Como se describió anteriormente, actualmente el *frontend* encargado de graficar los contextos genómicos obtiene la información necesaria desde archivos de texto plano que se encuentran en el servidor FTP de la NCBI, el cual contiene la base de de datos de genomas, archivos que para ser procesados y extraer los datos requeridos necesitan ser descargados y descomprimidos. Como claramente es un proceso ineficiente, la API que construiremos debe poder obtener estos datos desde un *backend* que no realice el mismo procedimiento, por lo que nos planteamos el uso de una base de datos en donde almacenaremos solo los datos necesarios de los genomas, los cuales obtendremos procesando los archivos de texto provenientes del servidor FTP y almacenarlos en nuestra base de datos, de forma que el proceso de obtener un genoma utilizando la API de GraphQL sea mucho más eficiente. Entonces para poder decidir inicialmente que tipo de base de datos vamos a utilizar, si una de tipo relacional o una no relacional primero debemos analizar cómo es la estructura de los archivos que contienen los genomas y de qué forma están presentes los datos requeridos por el frontend.

3.3.2. Estructura de un archivo de un genoma

Los archivos de la base de datos *Genome* que se encuentran en el servidor FTP del NCBI corresponden a archivos de texto plano con la extensión .gbff, extensión que nos dice que el archivo contiene todas las secuencias del genoma. Cada secuencia de un genoma tiene que seguir un formato[22] estricto definido por la NCBI, el cual define la cantidad de espacios que deben tener ciertas áreas, qué secciones debe contener, ítems opcionales y obligatorios , entre otras cosas. En este caso nos enfocaremos en analizar las secciones que contienen los datos de interés para este proyecto y que posteriormente extraeremos y almacenaremos en la base de datos.

Primero tenemos la información de cabecera, que se puede observar en la figura 10, la cual contiene datos importantes para nosotros como lo son:

- *DEFINITION*: definición de la secuencia del genoma.
- *ACCESSION*: el identificador único de la secuencia del genoma-

Además tenemos datos como:

- *LOCUS*: un nombre mnemónico para la secuencia, tiene que estar de forma obligatoria en el archivo.
- *VERSION*: un código compuesto por el *ACCESSION* y el número de versión asociada a la versión más actual de la secuencia, y al igual que las demás, es obligatoria.
- *DBLINK*: sección que contiene *links* hacia otras bases de datos que tienen relación con el genoma, y cuyo valor son una o más bases de datos, cada una seguida del identificador único del registro dentro de ellas.

```
LOCUS       NC_003074           23459830 bp    DNA     linear   CON 14-FEB-2019
DEFINITION  Arabidopsis thaliana chromosome 3 sequence.
ACCESSION   NC_003074
VERSION     NC_003074.8
DBLINK      BioProject: PRJNA116
            BioSample: SAMN03081427
            Assembly: GCF_000001735.4
```

Figura 10: Archivo de texto plano correspondiente a una secuencia de un genoma.
Fuente: *Arabidopsis thaliana chromosome 3 sequence. NCBI Nucleotide*.

Cada uno de estos datos, que forman parte de la cabecera del archivo deben cumplir que, la palabra clave, como por ejemplo *DEFINITION* o *LOCUS*, debe empezar en la columna 1,

[22] `https://ftp.ncbi.nih.gov/genbank/gbrel.txt`

mientras que el valor de cada palabra clave, es decir, lo que esta al lado derecho después de 1 o más espacios en blanco, debe comenzar en la columna 13 y puede ocupar hasta la columna 80 de esa línea en el archivo.

En la siguiente parte del archivo,que se puede ver en la figura 11, aún siendo parte de la cabecera, encontramos las referencias, indicada por la palabra clave *REFERENCE*, el cual presenta citaciones a publicaciones, revistas, autores, identificadores que referencian a otras bases de datos de publicaciones como PubMed o referencias a otras publicaciones de secuencias. Cada secuencia puede poseer múltiples referencias pero debe poseer al menos 1.

```
REFERENCE   1  (bases 1 to 23459830)
  AUTHORS   Salanoubat,M., Lemcke,K., Rieger,M., Ansorge,W., Unseld,M.,
            Fartmann,B., Valle,G., Blocker,H., Perez-Alonso,M., Obermaier,B.,
            Delseny,M., Boutry,M., Grivell,L.A., Mache,R., Puigdomenech,P., De
            Simone,V., Choisne,N., Artiguenave,F., Robert,C., Brottier,P.,
            Wincker,P., Cattolico,L., Weissenbach,J., Saurin,W., Quetier,F.,
            Schafer,M., Muller-Auer,S., Gabel,C., Fuchs,M., Benes,V.,
            Wurmbach,E., Drzonek,H., Erfle,H., Jordan,N., Bangert,S.,
            Wiedelmann,R., Kranz,H., Voss,H., Holland,R., Brandt,P.,
            Nyakatura,G., Vezzi,A., D'Angelo,M., Pallavicini,A., Toppo,S.,
            Simionati,B., Conrad,A., Hornischer,K., Kauer,G., Lohnert,T.H.,
            Nordsiek,G., Reichelt,J., Scharfe,M., Schon,O., Bargues,M.,
            Terol,J., Climent,J., Navarro,P., Collado,C., Perez-Perez,A.,
            Ottenwalder,B., Duchemin,D., Cooke,R., Laudie,M., Berger-Llauro,C.,
            Purnelle,B., Masuy,D., de Haan,M., Maarse,A.C., Alcaraz,J.P.,
            Cottet,A., Casacuberta,E., Monfort,A., Argiriou,A., flores,M.,
            Liguori,R., Vitale,D., Mannhaupt,G., Haase,D., Schoof,H., Rudd,S.,
            Zaccaria,P., Mewes,H.W., Mayer,K.F., Kaul,S., Town,C.D., Koo,H.L.,
            Tallon,L.J., Jenkins,J., Rooney,T., Rizzo,M., Walts,A.,
            Utterback,T., Fujii,C.Y., Shea,T.P., Creasy,T.H., Haas,B.,
            Maiti,R., Wu,D., Peterson,J., Van Aken,S., Pai,G., Militscher,J.,
            Sellers,P., Gill,J.E., Feldblyum,T.V., Preuss,D., Lin,X.,
            Nierman,W.C., Salzberg,S.L., White,O., Venter,J.C., Fraser,C.M.,
            Kaneko,T., Nakamura,Y., Sato,S., Kato,T., Asamizu,E., Sasamoto,S.,
            Kimura,T., Idesawa,K., Kawashima,K., Kishida,Y., Kiyokawa,C.,
            Kohara,M., Matsumoto,M., Matsuno,A., Muraki,A., Nakayama,S.,
            Nakazaki,N., Shinpo,S., Takeuchi,C., Wada,T., Watanabe,A.,
            Yamada,M., Yasuda,M. and Tabata,S.
  CONSRTM   European Union Chromosome 3 Arabidopsis Sequencing Consortium;
            Institute for Genomic Research; Kazusa DNA Research Institute
  TITLE     Sequence and analysis of chromosome 3 of the plant Arabidopsis
            thaliana
  JOURNAL   Nature 408 (6814), 820-822 (2000)
   PUBMED   11130713
REFERENCE   2  (bases 1 to 23459830)
  CONSRTM   NCBI Genome Project
  TITLE     Direct Submission
  JOURNAL   Submitted (14-FEB-2019) National Center for Biotechnology
            Information, NIH, Bethesda, MD 20894, USA
REFERENCE   3  (bases 1 to 23459830)
  AUTHORS   Krishnakumar,V., Cheng,C.-Y., Chan,A.P., Schobel,S., Kim,M.,
            Ferlanti,E.S., Belyaeva,I., Rosen,B.D., Micklem,G., Miller,J.R.,
            Vaughn,M. and Town,C.D.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAY-2016) Plant Genomics, J. Craig Venter Institute,
            9704 Medical Center Dr, Rockville, MD 20850, USA
  REMARK    Protein update by submitter
```

Figura 11: Archivo de secuencia de un genoma, sección de referencias.
Fuente: *Arabidopsis thaliana chromosome 3 sequence. NCBI Nucleotide.*

Las referencias pueden tener 7 sub palabras claves las cuales son:

- *AUTHORS*: Lista de los autores de la citación. Opcional.
- *CONSRTM*: Enumera los nombres colectivos de los consorcios asociados con la cita (ej: *International Human Genome Sequencing Consortium*), en lugar de los nombres individuales de los autores. Opcional.
- *TITLE*: Título completo de la cita. Opcional.
- *JOURNAL*: Indica el nombre de la revista, el volumen, el año y las páginas de la cita. Obligatorio, al menos 1 registro.
- *MEDLINE*: Proporciona el identificador único de Medline para una citación. Opcional.
- *PUBMED*: Proporciona el identificador único de PubMed para una cita. Opcional.
- *REMARK*: Especifica la relevancia de una cita para una entrada. Opcional.

Cada sub palabra clave, al igual que una palabra clave debe seguir un formato específico, y en este caso es que cada sub palabra clave debe comenzar en la columna 3 o 4 del archivo y su valor debe comenzar en la columna 13 y puede terminar en la columna 80.

Por último tenemos la sección más importante y que corresponde a la mayor parte del documento, los *FEATURES*, parte obligatoria de cada secuencia de un genoma y que como fue mencionado en la sección 3.2.1 posee distintas tipos de *features qualifiers* y al igual que las palabras claves del archivo, siguen un formato de clave-valor, donde cada clave corresponde al tipo de *feature* y su valor corresponde a información asociada a esta clave como lo es el locus_tag o el product, y cuyo primer valor siempre es la ubicación del *feature* dentro del gen, conocida como *feature location*, la cual está compuesta de una secuencia de números, símbolos (no letras) y funciones como:

- *complement* (ubicación): la característica es complementaria a la ubicación indicada.
- *join* (ubicación, ubicación, .. ubicación): Los elementos indicados deben colocarse de extremo a extremo para formar una secuencia contigua.
- *order* (ubicación, ubicación, .. ubicación): Los elementos se encuentran en el orden especificado.

Los *features* o características de una secuencia tienen un formato estricto dentro del archivo, donde cada *feature key* debe comenzar en la columna 6 y debe ser de a lo más 15 caracteres, mientras que los valores de cada clave, también conocidos como *Feature qualifiers*, deben comenzar en la columna 22 y pueden ocupar hasta la columna 80. Cada *Feature qualifier*, a excepción de la ubicación, debe seguir la siguiente estructura:

- El tipo de *qualifier* debe comenzar con el símbolo '/' seguido del tipo, como por ejemplo /product.
- Seguido de el tipo de *qualifier* debe venir el símbolo '=' representando que lo que esté después de este símbolo y entre comillas dobles '"' será el valor de este *qualifier*, entonces un *Feature qualifier* completo sería de la siguiente forma: /product="valor", como puede ser visto en la figura 12.

```
FEATURES             Location/Qualifiers
     source          1..23459830
                     /organism="Arabidopsis thaliana"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:3702"
                     /chromosome="3"
                     /ecotype="Columbia"
     gene            complement(1609..4159)
                     /locus_tag="AT3G01015"
                     /gene_synonym="TEL3N.1; TEL3N_1"
                     /db_xref="Araport:AT3G01015"
                     /db_xref="GeneID:821318"
                     /db_xref="TAIR:AT3G01015"
     mRNA            complement(join(1609..1936,2048..2142,2223..2282,
                     2428..2526,2690..2809,2885..2977,3064..3109,3203..4159))
                     /locus_tag="AT3G01015"
                     /gene_synonym="TEL3N.1; TEL3N_1"
                     /product="TPX2 (targeting protein for Xklp2) protein
                     family"
```

Figura 12: Archivo de secuencia de un genoma, sección de Features.
Fuente: *Arabidopsis thaliana chromosome 3 sequence. NCBI Nucleotide.*

Como se puede observar y entender de la estructura de los archivos de genomas, los datos están estructurados de una forma clave-valor donde cada palabra clave tiene su valor asignado a su lado derecho al igual que los *qualifiers* de cada *feature*, muy similar a lo que sería la estructura de un objeto JSON y a las consultas que se realizan utilizando el lenguaje GraphQL. Entonces el siguiente paso será definir qué base de datos es adecuada para almacenar este tipo de datos.

3.3.3. Definición de base de datos y modelo inicial

Para guardar los datos procesados provenientes de los archivos de genomas necesitamos escoger una base de datos que permita almacenarlos y luego consultarlos desde nuestra API de GraphQL, es aquí donde debemos analizar los datos descritos anteriormente y definir qué base de datos es adecuada. Primero tenemos los datos que se encuentran en la cabecera del archivo, que corresponden a *ACCESSION* y *DEFINITION*, los cuales siempre van a estar presentes y siguen la estructura definida anteriormente. Siendo el *ACCESSION* el identificador único de el archivo, la secuencia de un gen, ya tenemos la clave primaria para identificarlo dentro de cualquier base de datos. Después tenemos los *FEATURES*, la gran lista de características del genoma, estos están compuestos de ciertos calificadores, los cuales son todos de

carácter opcional dentro de cada *feature* a excepción de la ubicación. Los tipos de *features* a lo largo del genoma, como gen o mRNA, se pueden repetir indefinidamente y la única forma de identificar un *feature* de forma única es por medio del calificador *locus_tag* , el cual es de carácter opcional. Además, los calificadores que presenta cada tipo de *feature* dependen completamente de la secuencia del genoma en el que está, es decir, si tenemos un *feature* de tipo gen que presenta su identificador único, el locus tag, con ciertos calificadores como *protein* o *product*, puede que en otra secuencia de un genoma se encuentre el mismo *feature* con su *locus_tag*, pero no presente ninguno de los últimos 2 calificadores. Y al igual que los calificadores, la ubicación de cada *feature* depende completamente de la secuencia del genoma en la cual se encuentra.

Dado la estructura de los datos y la gran variabilidad que estos presentan debido a que la mayoría de los datos pueden o no estar presentes, necesitamos un esquema de datos flexible que permita almacenarlos, es por esto que decidimos trabajar con una base de datos no relacional, por lo que lo siguiente a decidir es cual utilizar de de entre la gran cantidad existente.

Ya que el implementar la solución óptima no está dentro del alcance de este trabajo, escogeremos una base de datos que cumpla con lo requerido por el proyecto y por lo tanto no se analizará extensivamente el rendimiento de cada base de datos no relacional existente. Por como está estructurado el archivo lo más natural a primera vista es usar una base de datos de tipo clave-valor o una de tipo documental que es una extensión de la primera. Ya que una base de datos clave-valor solo permite acceder a por medio de las claves, que en este caso serían los *accesssions* de cada secuencia de genoma, se desecha pues los tipos de consultas definidos anteriormente, además de consultar por un *accession* de un genoma también consultan mediante un locus_tag o un *assembly accession*, por lo que la opción por la que nos inclinamos es una base de datos documental.

Existen una gran cantidad de bases de datos documentales, estando entre las más utilizadas bases de datos como MongoDB, Amazon DynamoDB, Couchbase o Redis. Como la API estará disponible en un servidor, bases de datos *serverless*[23] como DynamoDB están descartadas, por lo que nos queda decidir entre el resto de bases de datos. En un estudio realizado por Tang et al.[Tang y Fan, 2016] donde compara el rendimiento de 5 bases de datos no relacionales, donde se encuentran 2 de las bases de datos documentales más usadas como MongoDB y Couchbase, podemos ver que MongoDB junto a Redis son una de las que presenta un mejor rendimiento cuando se trata de insertar y ejecutar operaciones como se pueden ver en la figura 13 , aunque su eficiencia con respecto a operaciones por segundo es superada por Couchbase cuando se trata de una cantidad muy grande de documentos. Ya que MongoDB presenta un buen rendimiento, su formato para guardar documentos es similar a JSON, lo que facilitará la creación de un modelo para almacenar los datos, y tiene una gran variedad de herramientas que nos pueden ayudar a desarrollar la API de manera más cómoda, la elección de la base de datos será MongoDB.

[23] Alta disponibilidad sin necesidad de que el consumidor mantenga el servidor.

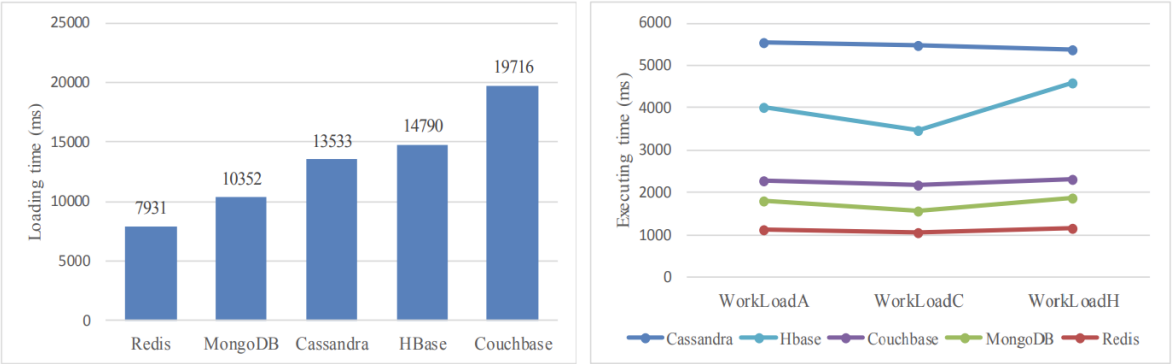

Figura 13: Tiempo de carga y ejecucón para 100.000 registros con 5 bases de datos.
Fuente: *Performance Comparison between Five NoSQL Databases . International Conference on Cloud Computing and Big Data (CCBD), 2016.*

Con la base de datos escogida, lo que necesitamos hacer ahora es crear un modelo que nos permita almacenar todos los datos requeridos por el *frontend* para que la API pueda obtenerlos, procesarlos y devolverlos. Como fue presentado anteriormente, MongoDB es una base de datos documental por lo que, de forma similar a una base de datos clave-valor, almacena documentos cuya estructura sigue un formato de clave-valor,como se puede ver en la figura 14.

```
{
  name: "sue",                    ← field: value
  age: 26,                        ← field: value
  status: "A",                    ← field: value
  groups: [ "news", "sports" ]    ← field: value
}
```

Figura 14: Ejemplo de documento en MongoDB.
Fuente: *Documentación de MongoDB. MongoDB Manual, Documents.*

Como la estructura de un archivo de una secuencia de genoma es muy similar, lo que hacemos para crear el modelo inicial es tomar las palabras claves del archivo de genoma, en este caso *DEFINITION*, *ACCESSION* y *FEATURES*, y utilizar estas como las claves en el documento de MongoDB, como se puede ver en la figura 15.

En este modelo el _id corresponde al *ACCESSION* de la secuencia del genoma, ya que este es su identificador único, las palabras clave *DEFINITION* y *FEATURES* pasaron a ser claves en el documento y mientras que el valor de *DEFINITION* es el mismo, el valor de *FEATURES* no puede ser simplemente un *string* ya que esta debe almacenar todas las características que presenta el genoma. Ya que las características se presentan como una lista, donde cada una presenta una clave como *gen* o *product*, las cuales se pueden ir repitiendo a lo largo

```
LOCUS       NC_003074           23459830 bp    DNA     linear   CON 14-FEB-2019
DEFINITION  Arabidopsis thaliana chromosome 3 sequence.
ACCESSION   NC_003074
VERSION     NC_003074.8
DBLINK      BioProject: PRJNA116
            BioSample: SAMN03081427
            Assembly: GCF_000001735.4
FEATURES             Location/Qualifiers
     source          1..23459830
                     /organism="Arabidopsis thaliana"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:3702"
                     /chromosome="3"
                     /ecotype="Columbia"
     gene            complement(1609..4159)
                     /locus_tag="AT3G01015"
                     /gene_synonym="TEL3N.1; TEL3N_1"
                     /db_xref="Araport:AT3G01015"
                     /db_xref="GeneID:821318"
                     /db_xref="TAIR:AT3G01015"
```

```
{
    "_id": "NC_003074",
    "definition": "Arabidopsis thaliana chromosome 3 sequence.",
    "features": ""
}
```

Figura 15: Modelo inicial para almacenar genomas.
Fuente: Elaboración propia.

del archivo y el único identificador único es el *locus_tag*, el cual es de carácter opcional, no tenemos algún elemento o clave que nos permita agrupar estas características, por lo que cada característica se almacenarán como un subdocumento, donde la clave será un campo de este subdocumento junto a sus calificadores, resultando en el modelo que se puede ver en la figura 16.

```
{
    "_id": "NC_003074",
    "definition": "Arabidopsos  thaliana chromosome 3 sequence.",
    "features":[
        {
            "location": "complement(1609..4159)",
            "key": "gene",
            "mobile_element_type": null,
            "locus_tag": "AT3FG01015"
            "gene": null,
            "product": null,
            "translation": null
        },
    ]
}
```

Figura 16: Modelo con *features* almacenados como subdocumentos embebidos.
Fuente: Elaboración propia.

El problema con este modelo es que la cantidad de características que presenta un genoma puede ser muy grande y en MongoDB existe un límite para el tamaño de los documentos que es de 16 megabytes, por lo que este límite puede ser superado por una gran cantidad de genomas que poseen una cantidad muy grande de características. Es por esto que cambiamos la forma de almacenar los subdocumentos de forma embebida a utilizar referencias, por lo que ahora cada característica corresponderá a un documento que se almacenará en una colección de *features*, donde tendrán un _id generado automáticamente por Mongo, a los cuales se les agrega el _id del genoma al que está asociado para poder realizar búsquedas desde una característica hacia el genoma. En el caso del documento que corresponde al genoma, el arreglo de *features* contendrá todos estos _id's de los *features*, que corresponden a referencias hacia cada característica, disminuyendo en gran medida el tamaño de estos documentos, resultando en un modelo como se puede ver en la figura 17.

```
//Colección de genomas
{
    "_id": "NC_003074",
    "definition": "Arabidopsos  thaliana chromosome 3 sequence.",
    "features":[
        ObjectId(**********0),
        ObjectId(**********1),
        .
    ]
}
//Colección de features
{
    "_id": ObjectId(**********0),
    "location": "complement(1609..4159)",
    "key": "gene",
    "mobile_element_type": null,
    "locus_tag": "AT3FG01015"
    "gene": null,
    "product": null,
    "translation": null,
    "genome_accession": "NC_003074"
}
```

Figura 17: Modelo con *features* almacenados en otra colección y utilizando referencias. Fuente: Elaboración propia.

Con este modelo ya se pueden almacenar los datos provenientes de los archivos de genomas por lo que el siguiente paso es buscar cómo este modelo puede integrar la información de las otras bases de datos utilizando GraphQL.

3.3.4. Esquema de GraphQL y mejora en el modelo de base de datos

Como ya sabemos, GraphQL nos permite consultar la información que uno quiera en cada query, utilizando su formato de consultas muy similar a JSON donde cada campo puede ser una consulta a distintas bases de datos, por lo que nos apoyaremos en esta funcionalidad para realizar la integración con las distintas bases de datos de la NCBI. Inicialmente diseñamos el esquema de GraphQL para consultar los genomas, el cual debe contener los campos *accession*, *definition* y la lista de*fetures*, por lo que el esquema es muy similar a lo que tenemos en el modelo de MongoDB, y al ser GraphQL un lenguaje con tipado fuerte a cada campo se le define su tipo como se puede observar en figura 18.

```
type Genome {
    _id: String
    definition: String
    features: [Feature!]!
}
```

Figura 18: Esquema de GraphQL para genomas.
Fuente: Elaboración propia.

En este caso el campo de *features* corresponde a un arreglo de múltiples *Feature*, donde *Feature* corresponde al esquema de las características del genoma. Este esquema contiene todos los campos que provienen del modelo de la base de datos Mongo, resultando en el esquema que se puede ver en la figura 19.

```
type Feature {
    _id: ObjectId!
    location: String!
    key: String
    mobile_element_type: String
    locus_tag: String
    gene: String
    product: String
    translation: String
    genome_accession: String!
}
```

Figura 19: Esquema de GraphQL para *features*.
Fuente: Elaboración propia.

Para integrar las demás bases de datos necesitamos algo que relacione el genoma con estas para así poder ir a buscar información relevante en las otras bases de datos de la NCBI com-

plementando la información que entrega nuestra API en relación a los genomas. Como se vio en la sección 3.3.2, el archivo de secuencias de un genoma presenta la palabra clave *DBLINK* cuyo valor contiene identificadores con referencia a otras bases de datos de la NCBI, como *Assembly*, *BioSample o BioProject* y además en la sección de referencias se pueden encontrar identificadores de la base de datos *PubMed*, teniendo así una referencia a todas las bases de datos que necesitamos integrar. En el caso de *Assembly*, *BioSample y BioProject*, corresponden a identificadores que podemos almacenar como un simple *string*, mientras que las referencias a la base de datos *PubMed* pueden ser muchas dependiendo de la cantidad de referencias que tenga el archivo, por lo que en este caso lo almacenaremos como un arreglo con todos estos identificadores, por lo que agregando los nuevos campos al modelo Mongo y al esquema de GraphQL Genome, tenemos el modelo y esquema actualizado de la figura 20.

```
{
    "_id": "NC_003074",
    "definition": "Arabidopsos  thaliana chromosome 3 sequence.",
    "assembly_link": "GCF_000001735.4",
    "bioproject_link": "PRJNA116",
    "biosample_link": "SAMN03081427",
    "pubmedIds":[
        "11130713",
    ],
    "features":[
        ObjectId(**********0),
        ObjectId(**********1),
        .
    ]
}
//Esquema de GraphQL
type Genome {
    _id: String!
    definition: String
    features: [Feature!]!
    gId: String!
    assembly_link: String
    bioproject_link: String
    biosample_link: String
    pubmedIds: [String!]
}
```

Figura 20: Modelo y esquema de genoma que incluye referencias a otras bases de datos. Fuente: Elaboración propia.

Ya que tenemos el modelo y esquemas base para comenzar con la implementación de la API lo siguiente será diseñar la arquitectura de la solución.

3.3.5. Arquitectura de la solución

La arquitectura de nuestra API contiene los *resolvers* de GraphQL que resolverán las consultas hacia la API, las cuales utilizarán un servicio para comunicarse con la base de datos Mongo donde tenemos almacenados todos los datos de genomas y para obtener la información de las otras bases de datos que se encuentran en la NCBI, como se puede ver en la figura 21.

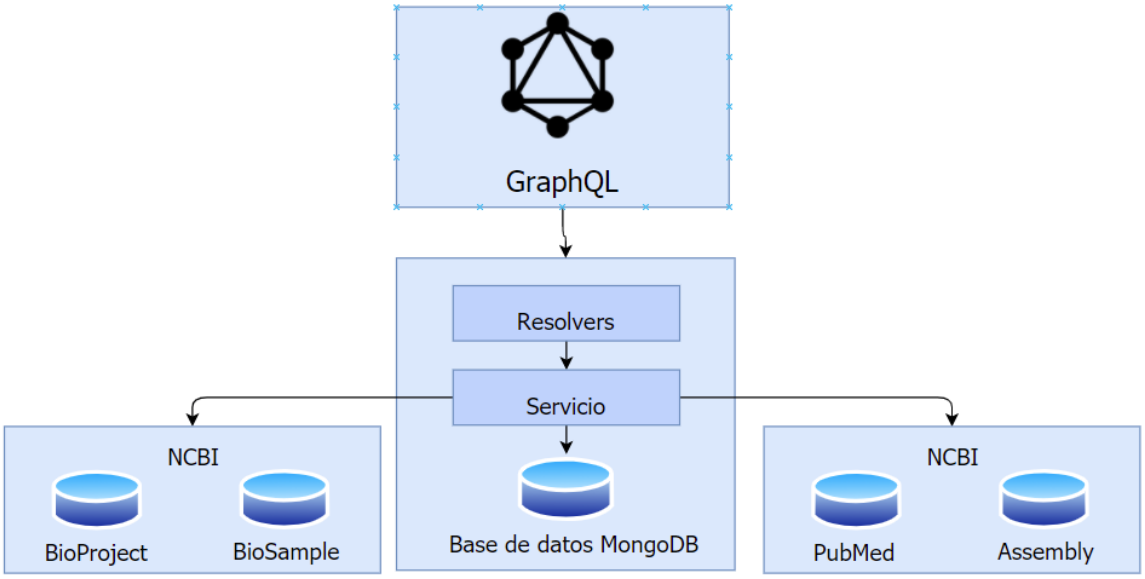

Figura 21: Arquitectura inicial de la API de GraphQL.
Fuente: Elaboración propia.

Además, para mejorar el tiempo de respuesta al obtener genomas desde la bases de datos Mongo, podemos implementar un caché al cual se le consulten inicialmente los genomas, disminuyendo los tiempos de respuesta si se necesitan datos del mismo genoma repetidamente y disminuyendo la carga con respecto a la cantidad de consultas que se realizan a la base de datos Mongo. Una de las bases de datos más utilizadas como caché en conjunto con una base de datos Mongo es Redis, un motor de bases de datos en memoria, donde podemos cachear los genomas utilizando como hash los identificadores de estos.

Un problema que surge con la cantidad de datos de genomas del servidor FTP de la NCBI es que es muy grande para ser almacenada localmente, por lo que la cantidad de datos almacenados inicialmente en Mongo será limitada y si se consulta un genoma que no se encuentra en la base de datos se tendrá que ir a buscar el genoma a la base de datos de la NCBI, procesarlo y guardarlo en la base de datos nuestra, esto implica que también debemos tener una

conexión a la base de datos *Genome*, por lo que la arquitectura de nuestra solución resulta en la figura 22.

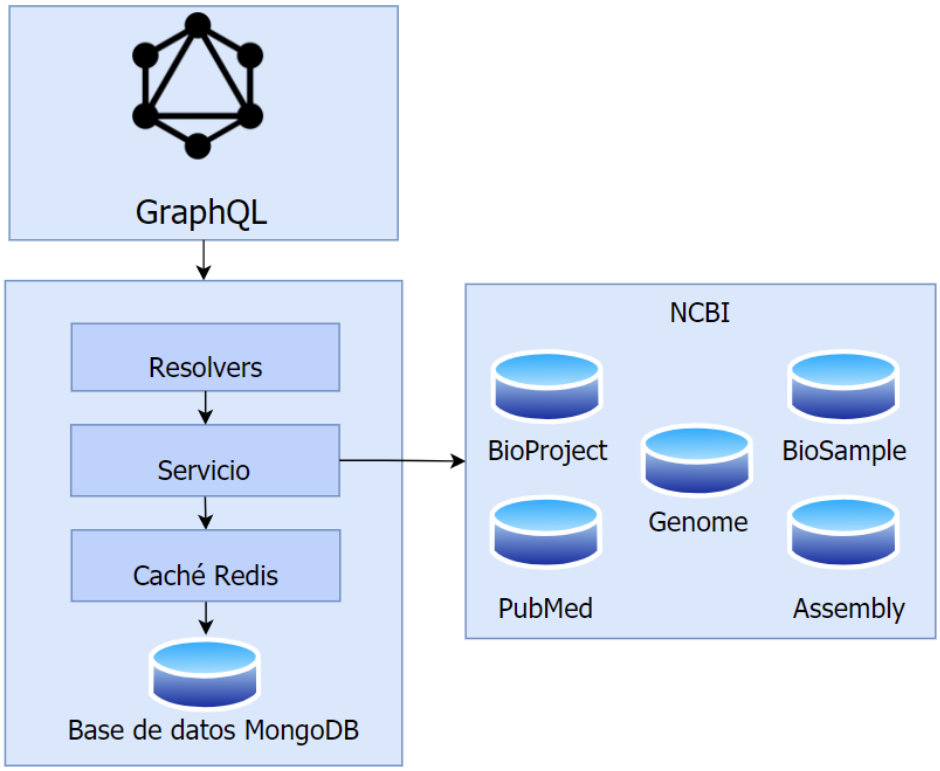

Figura 22: Arquitectura de la API con caché y conexión a la base de datos *Genome*.
Fuente: Elaboración propia.

3.4. Implementación de la solucion

Para la implementación de la API de GraphQL utilizaremos el lenguaje de programación TypeScript, un lenguaje fuertemente tipado basado en Javascript, lo cual nos permitirá desarrollar una solución más robusta al tener todo construido sobre un tipado fuerte por parte de los *resolvers* de GraphQL , los servicios, modelos y esquemas que serán implementados. Si bien hay otros lenguajes de tipado fuerte se escogió TypeScript por las herramientas que nos facilitan la construcción de una API de GraphQL como lo es *Apollo Server*[24], que utilizaremos para crear el servidor de GraphQL en base a un servidor de Express, el cual nos permite crear el servidor de manera rápida y flexible. La solución será implementada utilizando el *web framework* de Javascript más popular, Node.Js[25], sobre el cual construiremos la API de GraphQL utilizando las herramientas anteriormente mencionadas, además de otras necesa-

[24] `https://www.apollographql.com/docs/apollo-server/`
[25] `https://nodejs.org/es/`

rias para realizar las consultas a las bases de datos y la creación de modelos y esquemas que serán mencionadas más adelante.

Con respecto a inicializar la base de datos Mongo con los datos de los genomas provenientes de la NCBI necesitamos un *script* que procese los archivos de secuencias de genomas y los almacene en la base de datos siguiendo el modelo diseñado y para esto crearemos un *script* en el lenguaje de programación Python, un lenguaje liviano en el cual podemos implementar y ejecutar de manera rápida la inicialización de la base de datos.

3.4.1. Selección de datos iniciales

Para poblar la base de datos con los datos de genomas primero tenemos que definir qué datos vamos a escoger, ya que como se explicó anteriormente solo almacenaremos una cantidad limitada de datos. Los datos escogidos para almacenar inicialmente corresponden a un subconjunto de todos los datos de genomas llamado RefSeq, el cual es "la colección de secuencias de referencia (RefSeq) que proporciona un conjunto completo, integrado, no redundante y bien anotado de secuencias, incluyendo ADN genómico, transcritos y proteínas"[26], estos datos constituyen una base para estudios médicos ya que proporcionan "una referencia estable para la anotación del genoma, la identificación y caracterización de genes, el análisis de mutaciones y polimorfismos (especialmente los registros RefSeqGene), los estudios de expresión y los análisis comparativos", todo esto debido a que los genes de esta colección pasan por distintos procesos de revisión computacional y manual. Un archivo de texto plano con una lista de todos los genomas pertenecientes a RefSeq se puede encontrar en el servidor FTP de la NCBI dentro de la base de datos *Genome*, con el nombre de assembly_summary_refseq.txt el cual tiene una estructura que se puede ver en la figura 23, y es el que usaremos para encontrar los archivos de los genomas que procesaremos y almacenaremos en Mongo.

```
#   See ftp://ftp.ncbi.nlm.nih.gov/genomes/README_assembly_summary.txt
# assembly_accession    bioproject  biosample   wgs_master  refseq_cate
GCF_000001215.4 PRJNA164    SAMN02803731        reference genome    722
GCF_000001405.39    PRJNA168            reference genome    9606    966
GCF_000001635.27    PRJNA169            reference genome    10090   106
GCF_000001735.4 PRJNA116    SAMN03081427        reference genome    376
GCF_000001905.1 PRJNA70973  SAMN02953622    AAGU00000000.3  representat
GCF_000001985.1 PRJNA32665  SAMN02953685    ABAR00000000.1  representat
GCF_000002035.6 PRJNA13922  SAMN06930106        reference genome    795
```

Figura 23: Archivo de texto assembly_summary_refseq.
Fuente: Servidor FTP de la NCBI. Genome , RefSeq.

Este archivo tiene una estructura de tabla presentando cabeceras que nos indican qué tipo de información está bajo ellas y cada una de estas cabeceras están separadas entre sí por

[26] https://www.ncbi.nlm.nih.gov/refseq/about/

un tabulador, de igual forma que toda la información que viene debajo. En la figura 23 que muestra una parte del archivo podemos ver que la primera "columna" corresponde a él *assembly* accession de los genomas, la siguiente es *bioproject*, *biosample* y así en adelante, en este caso utilizaremos los *assembly accession* de cada genoma para buscar los archivos de genomas con sus secuencias que procesaremos y almacenaremos. Es aquí donde nos apoyaremos en las herramientas que nos entrega Entrez[27], específicamente la API de Entrez, API que nos permite buscar entre las bases de datos de la NCBI utilizando consultas https con cierto formato fijo a la cual le entregamos los parámetros para que realice las búsquedas.

3.4.2. Obtención de archivos de genomas mediante la API de Entrez

Para comenzar necesitamos saber en qué base de datos buscar los *assembly accessions*, si en la base de datos *Genome* , *Assembly* u otra. Revisando los resultados de búsqueda que nos entrega la NCBI en el buscador de su página web, que de igual forma utiliza Entrez para realizarlas, tenemos que la base de datos de nucleótidos *Nucleotide*, presenta los archivos de las secuencias de genomas separados individualmente, lo que nos facilita el trabajo de estar procesando los grandes archivos de la base de datos *Genome* que presentan todas las secuencias y por lo tanto hay demasiada información extra entre cada secuencia que no necesitamos y tendríamos que leer de igual forma, por lo que en vez de utilizar la base de datos *Genome* para obtener la información de los genomas utilizaremos la base de datos *Nucleotide* donde tenemos cada secuencia en un archivo individual, reemplazando en nuestra arquitectura la base de datos *Genome* por la de *Nucleotide* como se puede ver en la figura 24.

[27] `https://www.ncbi.nlm.nih.gov/books/NBK25497/`

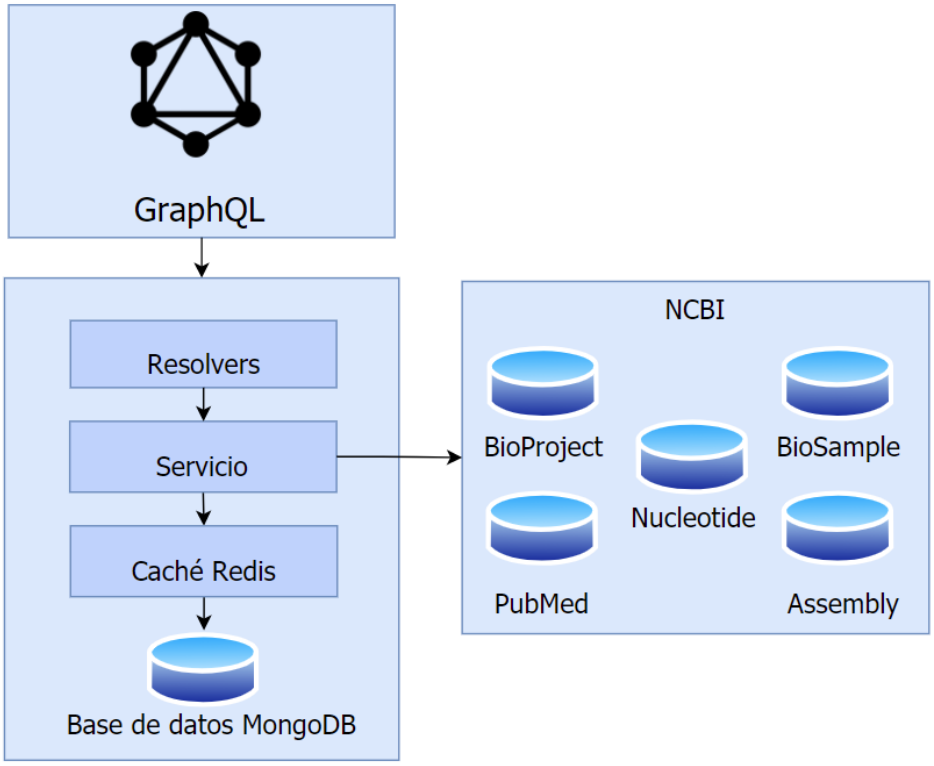

Figura 24: Arquitectura de la API actualizada con la base de datos *Nucleotide*.
Fuente: Elaboración propia.

Ya escogida la base de datos en la que realizaremos las búsquedas, utilizaremos la API Entrez para ejecutarlas y obtener los archivos de las secuencias de genomas, en este caso utilizaremos dos herramientas, ESearch para obtener el o los Ids únicos de cada secuencia de genoma utilizando como parámetros de búsqueda el *assembly accession*, y EFetch que utiliza estos Ids para buscar el archivo en la base de datos *Nucleotide* y nos permite descargarlo para poder procesarlo y almacenarlo. Primero tenemos la herramienta Esearch, la cual utiliza la URL fija:

```
https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi
```

y los parámetros de ***db*** para indicar en qué base de datos buscar, ***term*** para indicar el término que debe buscar y ***retmode*** para pedir que la respuesta se entrege en formato JSON que facilita la búsqueda de los Ids. Entonces un ejemplo de la búsqueda de Ids utilizando ESearch sería:

```
https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?
db=nuccore\&term\=GCF\_000001735.4[Assembly Accession]\&retmode=json
```

donde se indica que que la db es ***nuccore***, que es el nombre que utiliza Entrez para la base de datos *Nucleotide*, el término es el ***assembly accession*** GCF_000001735.4, además especificando entre los símbolos "[ ]" que el término corresponde a un *assembly accession* y el parámetro de ***retmode*** con valor JSON para el tipo de respuesta. Esta búsqueda nos entrega el resultado que se puede ver en la figura 25.


```
{
    "header":{
        "type":"esearch",
        "version":"0.3"
    },
    "esearchresult":{
        "count":"5",
        "retmax":"5",
        "retstart":"0",
        "idlist":["240256493",
                "240256243",
                "240255695",
                "240254678",
                "240254421"
                ],
        "translationset":[],
        "translationstack":[
            {
                "term":"GCF_000001735.4[Assembly]",
                "field":"Assembly",
                "count":"5",
                "explode":"N"
            },
            "GROUP"
        ],
        "querytranslation":"GCF_000001735.4[Assembly]"
    }
}
```


Figura 25: Resultado de una consulta utilizando la herramienta ESearch de Entrez.
Fuente: Elaboración propia.

Luego con los Ids que entrega la respuesta de ESearch, utilizamos EFetch para obtener los archivos individuales con la URL fija:

```
https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi
```

y al igual que ESearch los parámetros ***db*** y ***retmode*** para indicar la base de datos y el tipo de respuesta, el parámetro ***id*** para indicar el identificador del archivo que queremos obtener y un parámetro llamado ***rettype*** con el que indicaremos qué tipo de respuesta buscamos, en este caso el valor que utilizamos es ”gbwithparts” para indicar que queremos el archivo completo de la secuencia de un genoma.

Un ejemplo de EFetch sería:

```
https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=genome
&id=240256493&rettype=gbwithparts&retmode=text
```

donde indicamos que la db es nuccore, el id es uno de los obtenidos por la búsqueda de ESearch anterior, rettype tiene el valor de ”gbwithparts” como explicamos anteriormente y retmode con el valor de text para que el archivo a descargar sea uno de texto plano. El resultado de esta consulta es un archivo como el mostrado en las figuras 10, 11 y 12.

3.4.3. Inicialización de la base de datos

Ya teniendo el cómo obtener los archivos de secuencias de genomas, comenzaremos la creación del script de Python para procesar el archivo de *assembly_summary_refseq* e inicializar la base de datos con genomas.

Con cada *assembly accession* del arvhivo se descargan los arvhivos uno por uno y se procede a obtener todos los datos que necesitamos utilizando expresiones regulares, haciendo uso de las palabras claves y las columnas de inicio de cada dato como los *features keys* o los calificadores explicados en la sección 3.3.2, para identificar cada uno en las líneas del archivo. El documento de genoma se crea inicialmente como un diccionario de Python con una lista de *features* vacía a la cual se le van agregando los *ObjectIds* creados por Mongo cuando almacenemos los *features* en la base de datos, y además se agrega un campo adicional al modelo de Mongo que sería el identificador único que maneja la NCBI y la API de Entrez, la serie de números que nos entrega al consultar por una secuencia de genomas utilizando ESearch llamado *Sequence Identifier* y para el cual se utilizan las siglas “gi”, por lo que dentro

del modelo se llamará de igual forma “gi”. Además se crea un diccionario de Python vacío para almacenar los *features* que luego serán almacenados en Mongo cuando se obtenga toda la información de esta característica. Una vez el *feature* es almacenado en Mongo se obtiene el *ObjectId* generado para almacenarlo en la lista de *features* de el genoma y luego se vaciará el diccionario del *features* para poder almacenar el siguiente. El algoritmo expresado de forma general puede verse en la figura 26

```
re = r'^GCF\_\d*\.\d*'
FOR line in assemnbly_summary_refseq.txt:
    assembly = re.match(line)
    esearch_result = ESearch(assembly)
    FOR id in esearch_result.ids:
        genome = {gi: id , features: []}
        feature = {}
        fetch_response = EFetch(id)
        WRITE fetch_response in id.txt FILE
        FOR line in id.txt:
            #Obtener datos con expresiones regulares
            IF feature is complete:
            #Si se almacenaron todos los calificadores de el feature.
                #insertar en colección features de Mongo
                feature_id = featuresCollection.insert(feature)
                APPEND feature_id to genome[features]
                feature = {}
        #insertar genoma en colección de genomas de Mongo.
        genomesCollection.insert(genome)
        DELETE id.txt
        END
    END
END
```

Figura 26: Pseudocódigo del algoritmo de extracción y almacenamiento de datos de genomas.

Fuente: Elaboración propia.

Con respecto a las referencias provenientes de la palabra clave *DBLINK*, que contiene los identificadores de otras bases de datos relacionadas con el genoma, y que son necesarios para buscar información en las otras bases de datos, las utilizaremos para realizar búsquedas con la API de Entrez. Estas búsquedas se realizan con la herramienta ESearch para obtener el identificador que maneja Entrez para la referencia, el cuál se almacena en en un campo del modelo para que la API de GraphQL, al momento de resolver la consulta, evite realizar la búsqueda de ESearch nuevamente y utilice directamente el identificador almacenado para obtener la información de manera directa. Entonces cada vez que nuestro script se encuentre

con una referencia a otra base de datos en la palabra clave *DBLINK* haremos una búsqueda utilizando ESearch con la siguiente URL:

```
https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?
db={db}&term={link}&retmode=json
```

donde db es la base de datos correspondiente a cada link como assembly, bioproject o biosample y link es reemplazado por el identificador de cada una como por ejemplo **GCF_000001735.4** para assembly, **PRJNA116** para bioproject o S**AMN03081427** para biosample. En la figura 27 podemos ver el modelo final de Mongo para los genomas.

```
{
    "_id": "NC_003074",
    "definition": "Arabidopsos  thaliana chromosome 3 sequence.",
    "gI": "240255695",
    "assembly_link": "1733481",
    "bioproject_link": "116",
    "biosample_link": "3081427",
    "pubmedIds":[
        "11130713",
    ],
    "features":[
        ObjectId(**********0),
        ObjectId(**********1),
        .
    ]
}
```

Figura 27: Modelo de genoma que incluye el *Geninfo Identifier*(gi) y nuevas referencias utilizando los identificadores que maneja Entrez .
Fuente: Elaboración propia.

Una vez se almacenan los *features* en la base de datos solo queda almacenar el genoma que tiene las referencias a todos los documentos de *features* creados y, finalmente eliminar el archivo de texto descargado para así pasar al siguiente Id de la lista entregada por ESearch o pasar al siguiente *assembly accession*, si ya no se tienen más Ids.

3.4.4. Creación de modelos y esquemas

Ya que tenemos los datos almacenados en la base de datos Mongo, se tienen que crear los modelos y esquemas para que nuestra API pueda acceder a esta y obtenerlos. Para esta tarea y ya que estamos usando Typescript, nos apoyaremos en la biblioteca de JavaScript denominada Mongoose[28] que permite realizar la conexión de la API con la base de datos y Typegoose[29] junto a TypeGraphQL[30] para la creación de los modelos. Mongoose es una herramienta que permite crear de manera sencilla modelos, la cual incluye validación y construcción de consultas que permiten obtener documentos por medio de referencias de forma más rápida y eficiente, que es tal como tenemos construido nuestro modelo de datos, específicamente los *features*. Typegoose hace que la creación de modelos se pueda realizar mediante decoradores que aplican las mismas funcionalidades de Mongoose y, TypeGraphQL es una biblioteca que permite definir un esquema de GraphQL de manera rápida y simple utilizando decoradores y que a la vez se puede integrar fácilmente con otras bibliotecas que utilizan decoradores como Typegoose. Esta biblioteca también ayudará en la creación de los *resolvers*, *queries* y *field resolvers* de GraphQL. Los decoradores añaden las funcionalidades de forma dinámica a las clases que creamos, por lo que solo necesitamos crear una clase que contenga todos los campos que necesitamos para el modelo y esquema y aplicar los decoradores correspondientes a las propiedades que forman parte de cada uno, como se puede ver en la fig 28.

[28] `https://mongoosejs.com/`
[29] `https://typegoose.github.io/typegoose/`
[30] `https://typegraphql.com`

```
@ObjectType()
export class Genome {

    @prop()
    @Field()
    readonly _id!: String;

    @prop({ref: 'Feature'})
    @Field(_type => [Feature])
    features: Feature[];

    @prop()
    @Field({ nullable: true })
    definition?: String;
    .
    .
}
```

Figura 28: Definición de clase Genoma utilizando decoradores de Typegoose y TypeGraphQL. Fuente: Elaboración propia.

En este caso los campos que utiliza el decorador @prop formarán parte del modelo de Mongoose y los que utilizan el decorador de @Field formarán parte del esquema de GraphQL.

Además de crear todos los campos que posee el modelo y esquema, tenemos que crear campos adicionales para el esquema de GraphQL, los cuales se encargarán de tener toda la información proveniente de las otras bases de datos que integraremos, por lo que agregaremos los campos de *assembly info*, *bioproject_info* , *biosample_info y pubmed_info* al esquema de *Genome*, resultando en el esquema de la figura 29, y cada uno de estos campos tendrá su propio esquema de GraphQL con los campos que permitan almacenar la información proveniente de la NCBI.

```
type Genome {
    _id: String!
    features: [Feature!]!
    definition: String
    gId: String!
    assembly_link: String
    bioproject_link: String
    biosample_link: String
    pubmedIds: [String!]
    assembly_info: AssemblyInfo
    biosample_info: BiosampleInfo
    bioproject_info: BioprojectInfo
    pubmed_info: [PubmedInfo!]
}
```

Figura 29: Definición del esquema Genoma con los campos adicionales para la integración de datos.

Fuente: Elaboración propia.

Para obtener la información de las otras bases de datos, se utilizarán las herramientas de la API de Entrez, específicamente ESummary que nos entrega toda la metadata asociada a las publicaciones y archivos de las bases de datos *Assembly*, *PubMed*, *BioSample y BioProject*, y al igual que EFetch necesita los parámetros de ***db*** e ***id*** como se puede ver en la siguiente URL:

```
https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?
db={db}&id={id}&retmode=json
```

donde el parámetro db es una de las 4 bases de datos nombradas anteriormente e id corresponde a los ids de referencia que tenemos almacenados en los documentos de genomas.

Comenzaremos con la creación del esquema para la información proveniente de la base de datos *Assembly*, la información vamos a obtener de la consulta a la API de Entrez será la siguiente:

- *Assembly Accession*: Identificador único del *assembly*.
- *Latest Assembly Accession*: Último identificador registrado del Assembly.
- *Taxonomy Id*: identificador de un taxón en la base de datos de *Taxonomy* de la NCBI.

- *Species Name*: Nombre de la especie.
- *Specie Taxonomy Id*: Identificador taxonómico de la especie.
- *Submitter*: Organización que presentó el *genome assembly* para ser ingresado a la NCBI.
- *FTP Rpt*: Ruta en el servidor FTP de la NCBI donde se encuentra el reporte del assembly.

Todo esto es entregado en forma de un JSON gracias al parámetro de *retmode* que le entregamos a la consulta hecha con ESummary, por lo que solo se necesita hacer la consulta y obtener los datos utilizando las claves definidas en el objeto JSON, mientras que en nuestra API se almacenarán en el esquema de GraphQL *AssemblyInfo* de la figura 30.

```
type AssemblyInfo {
   assembly_accession: String
   latest_assembly_accession: String
   taxid: String
   specie: String
   specieTaxId: String
   submitter: String
   ftp_rpt: String
}
```

Figura 30: Esquema de GraphQL para la información proveniente de la base de datos *Assembly*.

Fuente: Elaboración propia.

Desde la base de datos *BioProject* obtendremos los siguientes datos:

- *Project Accession*: Identificador único del proyecto.
- Tipo de proyecto, en este caso existen 2 tipos, proyectos Umbrella que son de naturaleza administrativa y son creados a petición de un remitente, de una agencia de financiación o por el personal del NCBI para agrupar múltiples proyectos que forman parte de una gran iniciativa o colaboración o fuente de financiación o proyectos de "Presentación Primaria" que representan, y están vinculados a presentaciones de datos actuales o futuras. Las "Presentaciones Primarias"[31] incluyen una serie de atributos que describen la iniciativa y utilizan un vocabulario controlado.
- *Project Target Material*: indica el tipo de material que se aísla de la muestra para su uso en el estudio, este puede ser de los siguientes tipos:

[31] *Primary Submission*,

  - *Genome*: un genoma completo, también puede ser sólo el genoma nuclear[32].
  - *Purified chromosome*[33]: uno o más cromosomas o replicones[34] fueron purificados experimentalmente.
  - *Transcriptome*: datos de transcripción y/o expresión.
  - *Phenotype*[35]: datos descriptivos fenotípicos.
  - *Reagent*[36]: el material estudiado se obtuvo por reacción química.
  - *Proteome*[37]: datos de proteínas o péptidos
  - *Other*: se necesita especificar el material usado.

- *Project Target Scope*: indica el alcance y la pureza de la muestra biológica utilizada para el estudio, puede tener los siguientes valores:
  - *Monoisolate*: un solo animal, una línea celular cultivada, una población consanguínea, o cuando se necesitan múltiples individuos para recoger suficiente material y no se dispone de estos.
  - *Multiisolate*: varios individuos que representan colecciones de muestras distintas, una población representativa de una especie.
  - *Multi-species*: muestra representa varias especies.
  - textitEnvironment: el contenido de especies de la muestra no se conoce. Esto se utiliza para los estudios del metagenoma.
  - *Synthetic*: la muestra es sintetizada en un laboratorio.
  - *Other*: se necesita especificar el ámbito de la muestra que se ha utilizado.
- *Registration Date*: fecha de registro del proyecto.
- *Project Name*: nombre del proyecto.
- *Project Title*: Título del proyecto.
- *Project Description*: descripción del proyecto.
- *Organism name*: nombre del organismo que estudia el proyecto.
- *Organism strain*: cepa, o población a la que pertenece el organismo.
- *Organism label*: etiqueta del organismo que maneja la NCBI.

[32] El ADN nuclear es el material genético presente en el núcleo de cada célula en todos los seres vivos,
[33] Cromosomas que fueron purificados utilizando filtros para así poder usarlos en estudios y experimentosm
[34] Unidad de ADN o ARN en la cual ocurre un acto individual de replicación.
[35] Cualquier característica o rasgo observable de un organismo.
[36] Sustancia o compuesto añadido a un sistema para provocar una reacción química.
[37] Totalidad de proteínas expresadas en una célula particular bajo condiciones de medioambiente y etapa de desarrollo específicas.

- *Sequencing status*: estado de la secuenciación.
- *Submitter*: organización que presentó el proyecto.
- *supergroup*: supergrupo del organismo

También incluye una serie de claves relacionadas con la relevancia del proyecto respecto a ciertos ámbitos, aunque generalmente sus valores son *strings* vacíos.

- *Relevance Agricultural*.
- *Relevance Medical*.
- *Relevance Industrial*.
- *Relevance Environmentall*.
- *Relevance Evolution*.
- *Relevance Other*.
- *Relevance Model*.

Con estos campos generamos el esquema de *BioprojectInfo* de la figura 31.

Desde la base de datos *BioSample* obtendremos los siguientes datos:

- *Title*: título del documento que describe la muestra.
- *Accession*: Identificador único que se le asigna a la muestra presentada.
- *Publication Date*: fecha de publicación.
- *Organization*: organización que publicó los datos de la muestra.
- *Organism*: organismo del que corresponden los datos de la muestra.

formando con ellos el esquema *BioSampleInfo* que se puede ver en la figura 32.

Por último, tenemos los datos provenientes de las publicaciones en la base de datos PubMed:

- *Pub Date*: fecha de publicación.
- *Source*: fuente de la publicación
- *Title*: título de la publicación.

```
type BioprojectInfo {
   project_accession: String
   project_type: String
   project_target_material: String
   project_target_scope: String
   registration_date: String
   project_name: String
   project_title: String
   project_description: String
   relevance_agricultural: String
   relevance_medical: String
   relevance_industrial: String
   relevance_environmental: String
   relevance_evolution: String
   relevance_other: String
   relevance_model: String
   organism_name: String
   organism_strain: String
   organism_label: String
   sequencing_status: String
   submitter: String
   supergroup: String
}
```

Figura 31: Esquema de GraphQL para la información proveniente de la base de datos *BioProject*.

Fuente: Elaboración propia.

- *ISSN*: número internacional que permite identificar de manera única la publicación.
- *ESSN* al igual que el ISSN es un identificador, pero solo puede asignarse si existe el registro en la revista *NLM Catalog*.
- *Pub Type*: tipo de publicación.
- *Record Status*: estado del registro, si esta en proceso para ser registrado en *PubMed* o *MEDLINE*, o si ya esta indexado.

Con estos campos generamos el esquema *PubMedInfo* de la figura 33. Si en el futuro se agregan más campos de interés a la información entregada por la API de Entrez o se quiere obtener más de los ya existentes, los esquemas se pueden modificar fácilmente agregando el nuevo campo y accediendo a la clave correspondiente en el JSON de respuesta obtenido de la búsqueda con ESummary.

```
type BiosampleInfo {
   title: String
   accession: String
   publication_date: String
   organization: String
   organism: String
}
```

Figura 32: Esquema de GraphQL para la información proveniente de la base de datos *BioSample*.

Fuente: Elaboración propia.

```
type PubmedInfo {
   pub_date: String
   source: String
   title: String
   issn: String
   essn: String
   pub_type: String
   record_status: String
}
```

Figura 33: Esquema de GraphQL para la información proveniente de la base de datos *Pubmed*.

Fuente: Elaboración propia.

3.4.5. Creación de servicio y los *resolvers* de GraphQL

Ya teniendo los modelos y esquemas creados para poder obtener la información desde las bases de datos y poder generar los objetos de respuesta que entregará nuestra API, lo que se necesita hacer es crear el servicio y los *resolvers* para responder a las consultas que realizará el *frontend*, definidas en la sección 3.2.2,para las cuales utilizaremos la biblioteca de TypeGraphQL presentada anteriormente. El flujo para resolver una consulta hecha a nuestra API será de la siguiente manera:

- La consulta primero llegará al *resolver* de nuestra API, el que se encarga de recibir la consulta, los parámetros y revisar los errores que puedan provenir de estos.
- Este *resolver* necesita obtener el genoma pedido, para lo cual utiliza el servicio, que presenta la lógica de negocio de nuestra aplicación, el cual nos debe devolver el genoma o el correspondiente error, dependiendo si el genoma no existe, si existe pero no

se encuentra en nuestra base de datos o, si está siendo procesado.

- El servicio que necesita entregar el genoma al *resolver* debe comunicarse con la base de datos, para lo que utilizamos el modelo definido con Mongoose y realizamos la consulta directa a la base de datos, llenando las referencias necesarias de los *features*.
- Una vez el modelo devuelve el genoma al servicio, este último puede devolverlo al *resolver*, en caso de que no exista en la base de datos, el servicio deberá revisar en la base de datos *Nucleotide* de la NCBI si existe o no. En caso de que exista se comenzará la búsqueda y procesamiento del archivo utilizando un script de Python basado en el usado para inicializar la base de datos, devolviendo al usuario un error describiendo que está siendo procesado. En el caso de que el genoma no exista en la NCBI, simplemente se devuelve el error de *not found* al *resolver*.
- Finalmente, el *resolver* tendrá que entregar el genoma pedido en la consulta o el error correspondiente. En caso de que el genoma exista, se resolverán los campos de *AssemblyInfo* , *BioSampleInfo*, *BioProjectInfo y PubMedInfo* de forma paralela, esto gracias a los *Field resolvers* de GraphQL que podemos definir simplemente creando una función y adhiriendo el decorador de TypeGraphQL correspondiente. Cada uno de estos *field resolver* se comunica con el servicio de nuestra aplicación, el cual se encarga de realizar las consultas necesarias a la API de Entrez utilizando ESummary.

Entonces tenemos la primera consulta que requiere los parámetros de un *assembly accession*, un *locus tag* de inicio y uno de final, para así entregar todos los *features* que se encuentren entre estos dos *locus tag*. Todos los parámetros de esta consulta son obligatorios por lo que la falta de uno resultará en un error, el cual está manejado automáticamente por el *resolver*, por lo que los errores que manejamos manualmente es en el caso de que el valor de algún parámetro sea un *string* vacío. Para esta consulta necesitamos obligatoriamente el *accession assembly* y al menos un *locus tag*, por lo que si esta condición no se cumple se entrega el error de que existe algún parámetro que no puede ser *null*. Ya que el *locus tag* buscado puede estar en cualquiera de los archivos de secuencias de genomas que pertenecen a un único *assembly accession* y no tenemos este dato almacenado en el modelo de el genoma, la forma en que hacemos la búsqueda es inicialmente obtener el Id del *accession assembly* utilizando la herramienta ESearch de Entrez y con este Id buscar los códigos de proyectos relacionados al *assembly*. Luego con estos codigos, el *assembly accession* y el o los *locus tag* realizamos una búsqueda con ESearch en la base de datos *Nucleotide* entregándonos el Id exacto de la secuencia de genoma que contiene el o los *locus tags*, o en caso de que no exista simplemente devolvemos el error correspondiente. Con este Id podemos buscar en nuestra base de datos mediante el campo **gi** y si existe devolverlo al *resolver* o en caso de que no exista en nuestra base de datos ejecutar el script de Python para procesar y almacenar el genoma mientras que devolvemos el error de “genoma no encontrado en la base de datos , pero está siendo procesado” al usuario. Luego de que el genoma es devuelto al *resolver* se divide la lista de *features* tomando como inicio el *locus tag* de inicio y final el *locus tag* de final, en caso de que el *locus tag* de inicio sea nulo y el de final no, se asigna por

defecto que se dividirán los *features* desde el primero que exista hasta el *locus tag* de final definido por el parámetro mientras que si el *locus tag* de final es nulo y el de inicio no, la lista de *features* se dividirá desde el *locus tag* de inicio hasta el último *feature* existente en el genoma.

Para el segundo tipo de consulta se requieren los parámetros de un *locus tag* y 2 parámetros numéricos que corresponden a un límite superior e inferior, siendo el límite superior cuántos *features* se quieren obtener después del *locus tag* buscado y el límite inferior la cantidad de features anteriores al *locus tag* buscado, es decir, si se tiene un límite inferior de 2 y uno superior de 3 entonces se quieren obtener los 2 *features* anteriores al *locus tag* buscado y los 3 *features* siguientes al *locus tag*, incluyéndolo. Además se tiene un parámetro opcional que corresponde a un *assembly accession* en caso de que se quiera especificar un *assembly* en específico. Para resolver esta consulta y al igual que la anterior, se manejan manualmente los errores de parámetros nulos y se tienen 2 opciones de búsqueda:

- Si se entrega el parámetro de *assembly accession* se realiza la misma búsqueda en la API de Entrez con este parámetro y el *locus tag* para luego buscar en nuestra base de datos y devolver el genoma, el error correspondiente o ejecutar el script para almacenar un genoma nuevo.
- Si solo se entrega el *locus tag* y los límites inferior y superior, llamamos a otra función del servicio que en este caso busca en la colección de *features* el locus tag, si no se encuentra el locus tag en ningún feature de la base de datos se utiliza la API de Entrez para buscar el *locus tag* en la base de datos *Nucleotide* usando ESearch, se escoge el primer Id que contenga el *locus tag* y se almacena en la base de datos. En caso de que se encuentre el *locus tag* en nuestra colección de *features* utilizamos el campo de *genome_accession* para obtener el genoma asociado y devolverlo al *resolver*.

Cuando el *resolver* obtiene el genoma se utilizan los parámetros de los límites inferior y superior para dividir la lista de *features* y devolver los pedidos por el usuario. En caso de que no se encuentre el gen se pueden devolver los mismo errores que se presentaron en el primer tipo de consulta.

Por último, tenemos una consulta que no está dentro de las requeridas, una consulta opcional que utiliza los identificadores únicos de los genomas, el *accession number* de una secuencia de genoma, el que se encuentra en cada archivo y que corresponde al campo **_id** de nuestro modelo. Ya que la búsqueda más básica y más eficiente es la que utiliza los ids, también fue implementada en adición a las 2 tipos de consultas pedidas. Esta consulta pide como parámetros el *accession* del genoma, un *locus tag* de inicio y uno de final al igual que la primera consulta y esta es resuelta simplemente consultando por el **_id**, el *accession* del genoma, buscando en la base de datos nuestra inicialmente y devolviendo el genoma si es que existe o creando uno nuevo, si es que no existe pero se encuentra dentro de la NCBI.

3.4.6. Obtención de nuevos genomas

Como se habló en la sección anterior, al consultar un genoma que no existe en nuestra base de datos pero existe en la base de datos de la NCBI, lo que necesitamos es procesar y almacenar este nuevo genoma, para ello utilizamos un *script* de Python muy similar al usado para extraer y almacenar los genomas desde el archivo que contiene los *assembly accession* de refseq. Este *script* recibe como parámetro el id de una secuencia de genoma, también conocido como *Geninfo Identifier* o **gi** como lo almacenamos en nuestra base de datos, que como ya sabemos son los ids que maneja Entrez para realizar las búsquedas. Con este id el *script* simplemente descarga el genoma utilizando EFetch y lo procesa de igual forma que el *script* utilizado para inicializar la base de datos. El pseudocódigo describiendo de forma general el algoritmo se puede ver en la figura 34.

```
CrearGenoma(_id):
    genome = {gi: _id , features: []}
    feature = {}
    fetch_response = EFetch(_id)
    WRITE fetch_response in _id.txt FILE
    FOR line in _id.txt:
        #Obtener datos con expresiones regulares
        IF feature is complete:
        #Si se almacenaron todos los calificadores de el feature.
            #insertar en colección features de Mongo
            feature_id = featuresCollection.insert(feature)
            APPEND feature_id to genome[features]
            feature = {}
    #insertar genoma en colección de genomas de Mongo.
    genomesCollection.insert(genome)
    DELETE _id.txt
    END
```

Figura 34: Pseudocódigo del algoritmo de obtención y almacenamiento de un nuevo genoma. Fuente: Elaboración propia.

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

Para validar la solución se realizaron distintas pruebas que permiten comparar el rendimiento de nuestra solución con respecto a los distintos tipos de consultas, teniendo en cuenta el tamaño de los documentos y qué información se le pide a la API. Estas fueron realizadas en un computador con las siguientes características:

- Procesador Intel core I5-8250U 1.60GHz-1.80 GHz.
- 8 GB de memoria RAM.
- Sistema operativo Windows 10 Pro.

Y la aplicación junto con la base de datos Mongodb y Redis fueron construidas dentro de 3 contenedores Docker separados, ambiente que tiene a disposición 4 GB de memoria RAM.

El rango de tamaño entre los documentos de genomas almacenados en la base de datos Mongo van desde los 200 Bytes hasta 1 Megabytes, teniendo referencias a features desde 1 hasta más de 40.000 referencias, mientras que los tamaños de las referencias pueden ir mayormente desde los 100 Bytes hasta los 1.000 Bytes, teniendo muy pocos casos donde el tamaño de un feature puede llegar a los 5.000 Bytes.

4.1. Resultados de consultas para documento grande

El documento más grande encontrado en nuestra base de datos corresponde al genoma NC_003282, el cual tiene un tamaño de 876.811 Bytes, posee 46.720 *features* y 1 publicación referenciada en PubMed. Con este documento se realizaron las 3 tipos de consultas existentes en nuestra API, midiendo los tiempos de respuesta al obtener los datos de cada base de datos integrada.

Primero tenemos la consulta utilizando el *accession* del gen:

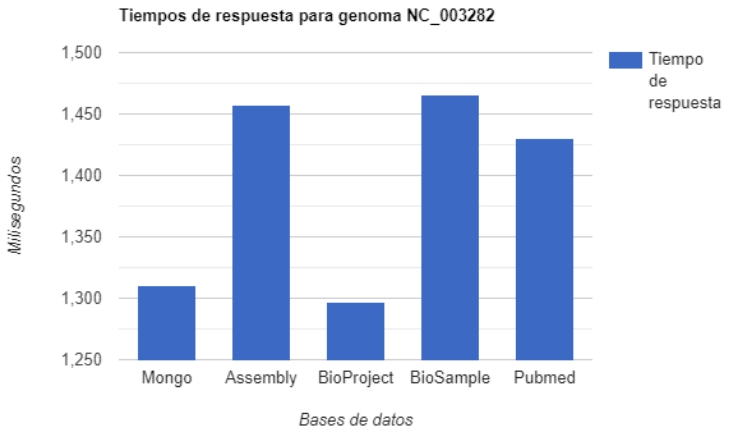

Figura 35: Tiempos de respuesta para genoma NC_003282 buscando por *genome accession*.
Fuente: Elaboración propia.

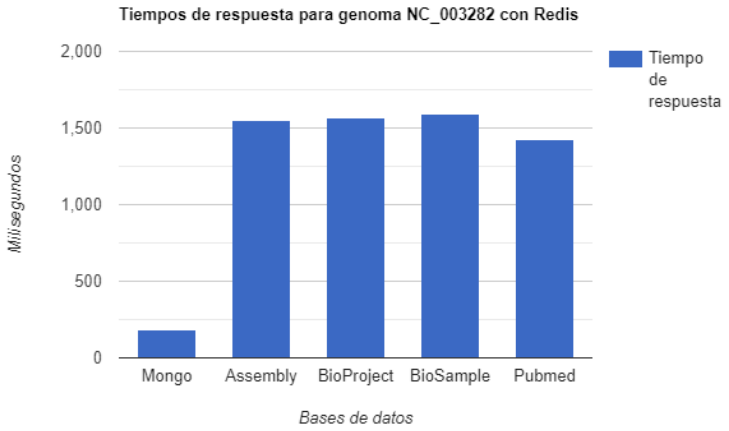

Figura 36: Tiempos de respuesta para genoma NC_003282 utilizando caché y buscando por *genome accession*.
Fuente: Elaboración propia.

donde tenemos tiempos de entre 1 y 2 segundos para obtener los distintos campos de información que provienen desde distintas bases de datos. Utilizando Redis como caché para los genomas tenemos que el tiempo de obtención del mismo se reduce de los 1300 milisegundos a solo 181 , una gran mejora en el rendimiento si se necesita obtener únicamente el genoma.

Con la consulta realizada utilizando un *assembly accession* tenemos los siguientes tiempos con y sin la utilización del caché de Redis:

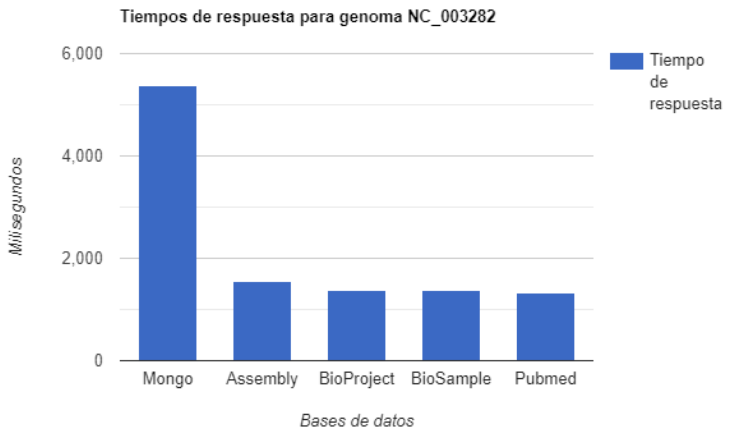

Figura 37: Tiempos de respuesta para genoma NC_003282 buscando por *assembly accession*.
Fuente: Elaboración propia.

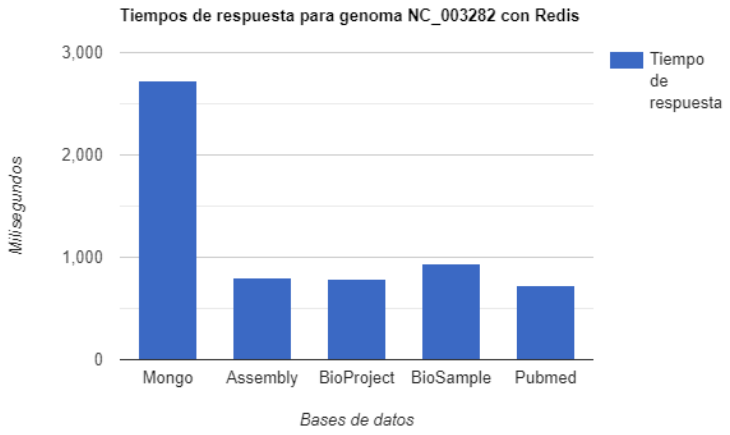

Figura 38: Tiempos de respuesta para genoma NC_003282 utilizando caché y buscando por *assembly accession*.
Fuente: Elaboración propia.

donde claramente el tiempo de obtención del genoma es mucho mayor debido a la cantidad

de consultas que hace a la API de Entrez para asegurar que se obtiene el genoma correcto.

Por último tenemos los tiempos de respuesta para la consulta utilizando un *locus tag*:

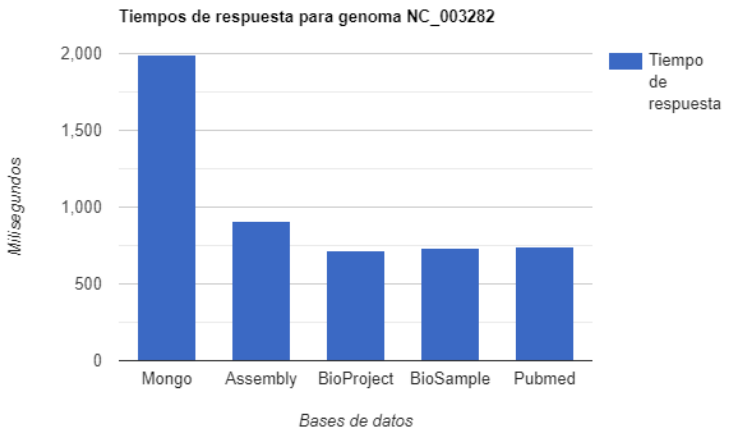

Figura 39: Tiempos de respuesta para genoma NC_003282 buscando por *locus tag*.
Fuente: Elaboración propia.

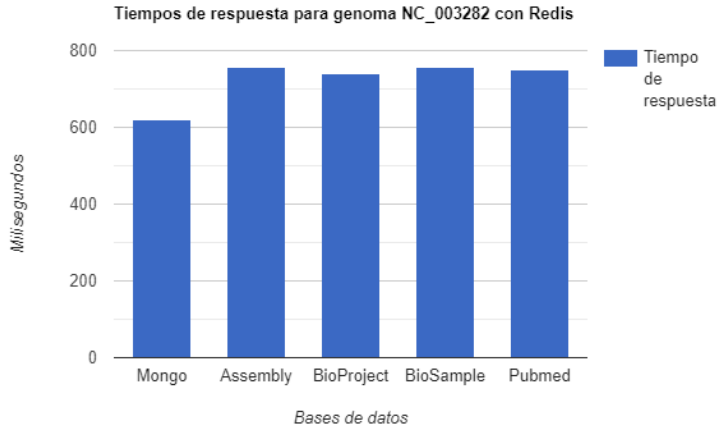

Figura 40: Tiempos de respuesta para genoma NC_003282 utilizando caché y buscando por *locus tag*.

Fuente: Elaboración propia.

donde volvemos a tener menores tiempos de obtención de genoma y menores tiempos de obtención de información proveniente de otras bases de datos, aunque los últimos dependen del tiempo de respuesta de la API de Entrez.

4.2. Resultados de consultas para documento de tamaño mediano

En este caso se realizaron las mismas consultas que para el documento anterior, pero utilizando el genoma NT_037436 que tiene un tamaño de 317.918 Bytes, 17.294 referencias a features y 11 publicaciones relacionadas en PubMed. Consultando por *accession* tenemos los siguientes resultados:

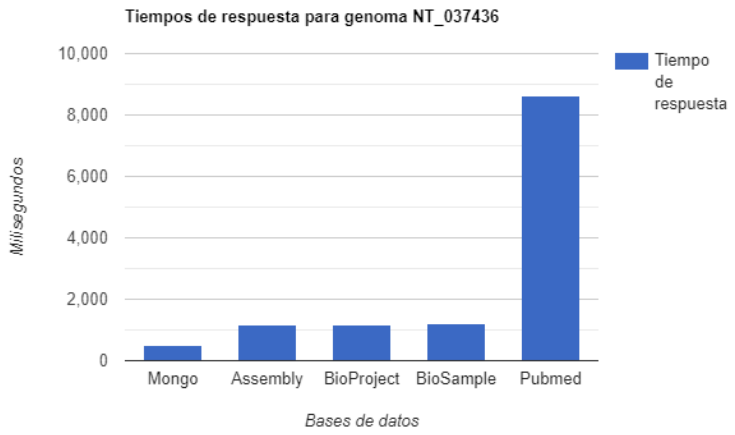

Figura 41: Tiempos de respuesta para genoma NT_037436 buscando por *genome accession*.
Fuente: Elaboración propia.

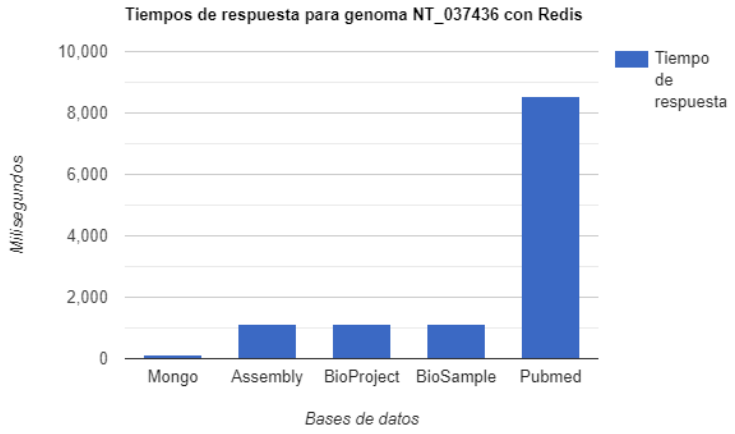

Figura 42: Tiempos de respuesta para genoma NT_037436 utilizando caché y buscando por *genome accession*.
Fuente: Elaboración propia.

donde vemos que obtener el genoma toma menos de 1 segundo, mientras que la obtención de información proveniente de las otras bases de datos se mantiene entre los 1 y 2 segundos

a excepción de las publicaciones de PubMed cuyo tiempo de respuesta es cercano a los 8 segundos, lo que se debe a que este genoma presenta referencia a 11 publicaciones y el tiempo de obtención de estas va aumentando proporcionalmente a la cantidad de publicaciones que se necesitan consultar a esta base de datos.

Consultando por *assembly accession*, tenemos los siguientes tiempos de respuesta:

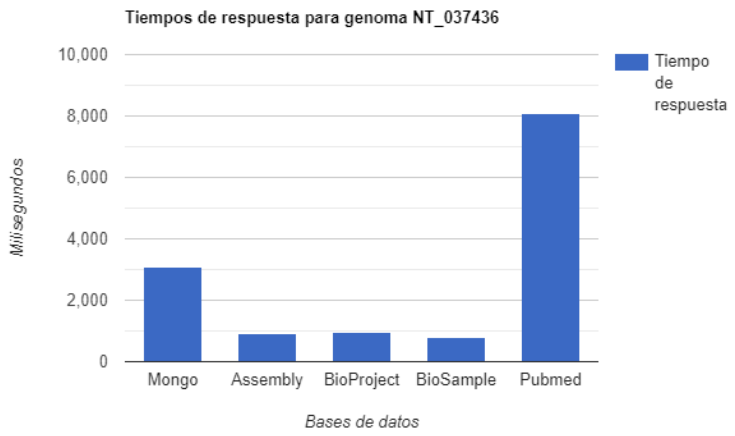

Figura 43: Tiempos de respuesta para genoma NT_037436 buscando por *assembly accession*.
Fuente: Elaboración propia.

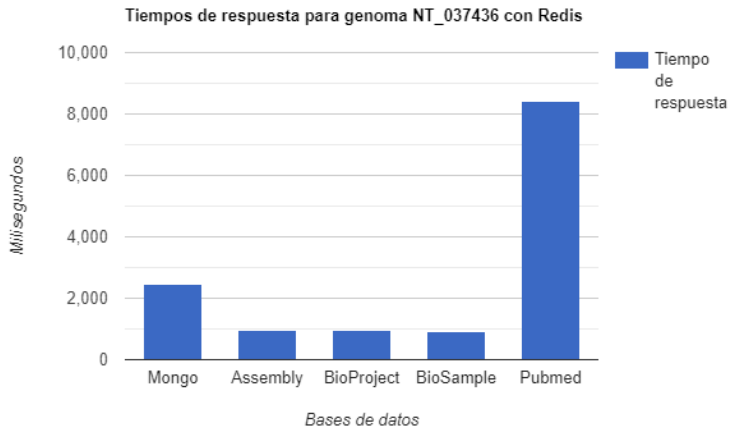

Figura 44: Tiempos de respuesta para genoma NT_037436 utilizando caché y buscando por *assembly accession*.
Fuente: Elaboración propia.

Y por último los tiempos de respuesta utilizando la búsqueda por *locus tag*:

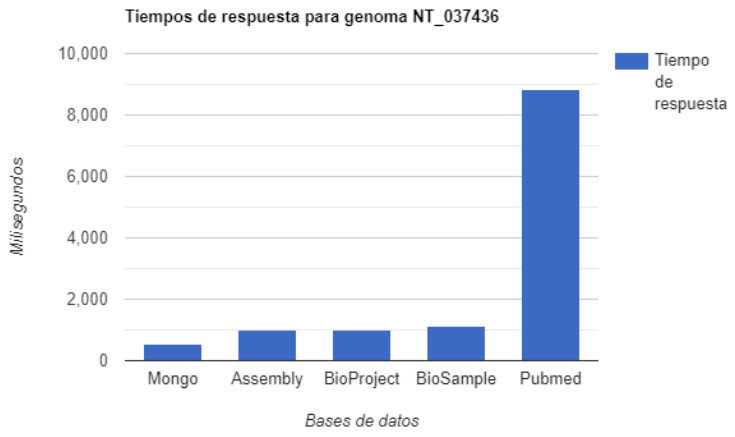

Figura 45: Tiempos de respuesta para genoma NT_037436 buscando por *locus tag*.
Fuente: Elaboración propia.

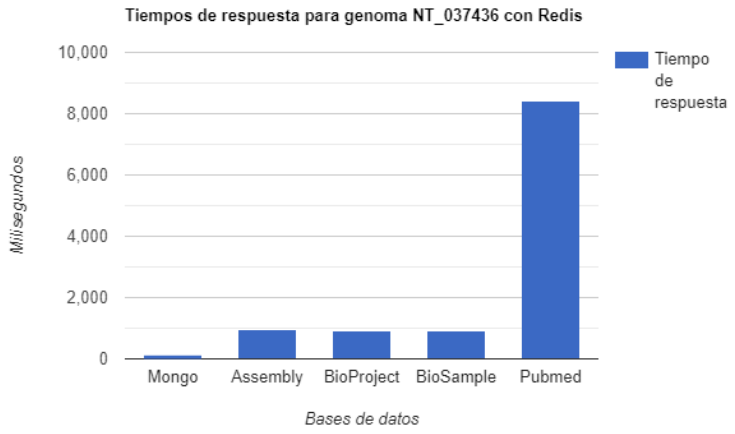

Figura 46: Tiempos de respuesta para genoma NT_037436 utilizando caché y buscando por *locus tag*.
Fuente: Elaboración propia.

con resultados similares a las otras consultas, donde la obtención de las publicaciones de PubMed es el proceso más largo, pero el tiempo de respuesta al obtener el genoma es de menos de 1 segundo debido a su tamaño.

4.3. Resultados de consultas para documento pequeño

Para estas consultas se utilizó el genoma NW_007931121 que tiene un tamaño de 1.146 Bytes, presenta 11 referencias a publicaciones y tiene 46 features. Se realizaron las mismas 3 tipos de consultas, comenzado por la búsqueda por *accession* y obteniendo los siguientes resultados:

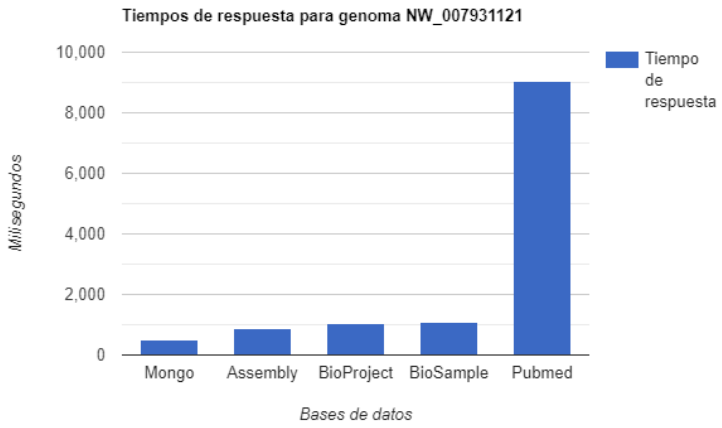

Figura 47: Tiempos de respuesta para genoma NW_007931121 buscando por *genome accession*.

Fuente: Elaboración propia.

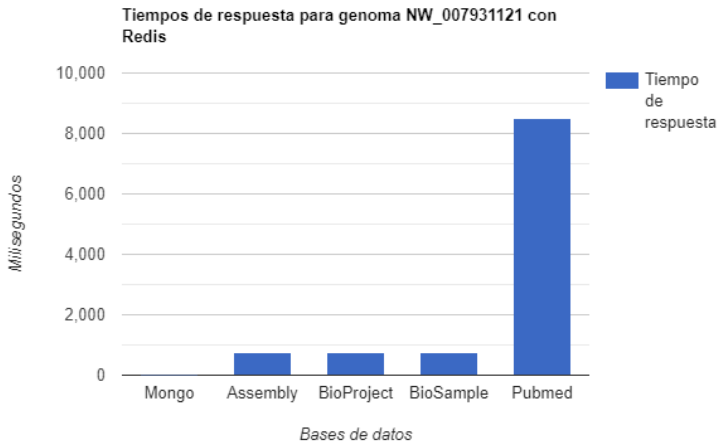

Figura 48: Tiempos de respuesta para genoma NW_007931121 utilizando caché y buscando por *genome accession*.

Fuente: Elaboración propia.

con tiempos muy similares a la consulta por *accession* del documento de tamaño medio,

donde la diferencia se puede ver al utilizar Redis en el documento pequeño teniendo un tiempo de respuesta muy cercano a los 0 segundos.

Realizando la búsqueda por *assembly accession* tenemos los siguientes resultados:

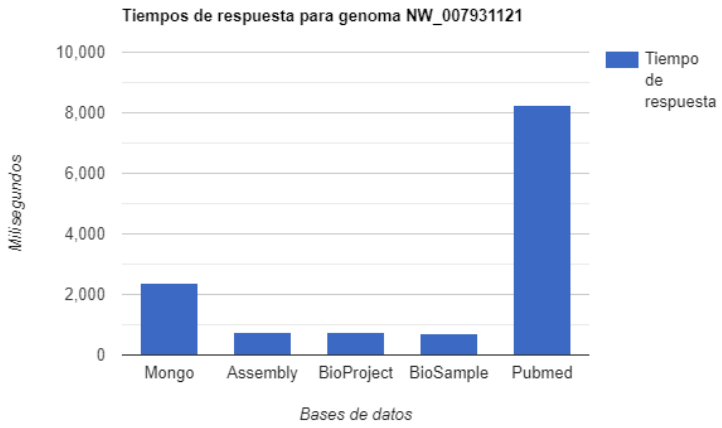

Figura 49: Tiempos de respuesta para genoma NW_007931121 buscando por *assembly accession*.

Fuente: Elaboración propia.

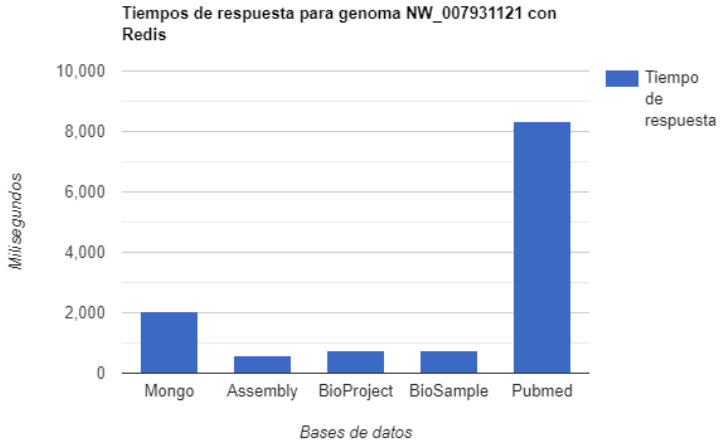

Figura 50: Tiempos de respuesta para genoma NW_007931121 utilizando caché y buscando por *assembly accession*.

Fuente: Elaboración propia.

Y por último las consultas buscando por *locus tag*:

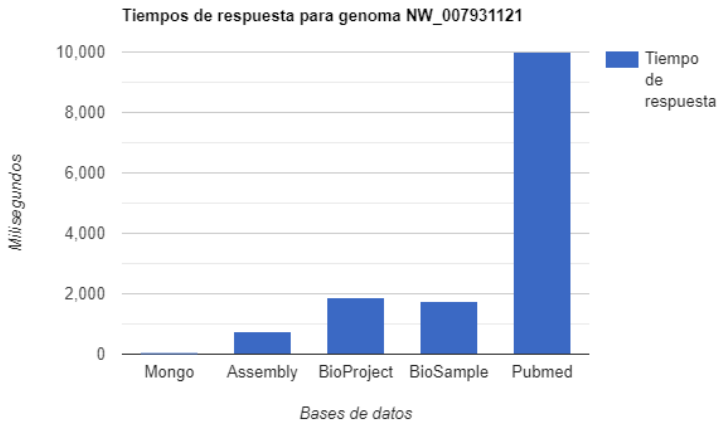

Figura 51: Tiempos de respuesta para genoma NW_007931121 buscando por *locus tag*.
Fuente: Elaboración propia.

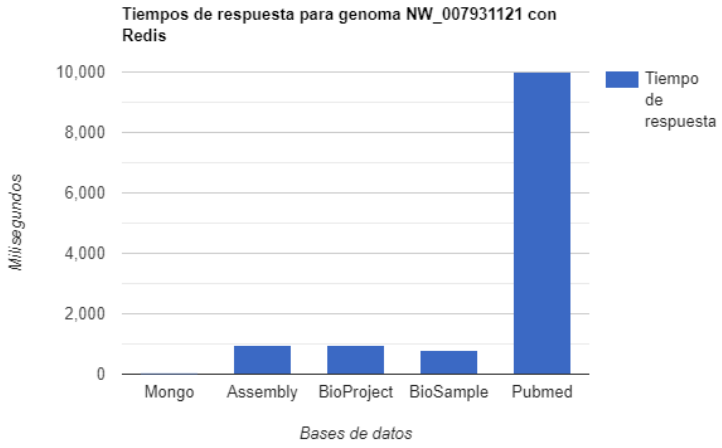

Figura 52: Tiempos de respuesta para genoma NW_007931121 utilizando caché y buscando por *locus tag*.

Fuente: Elaboración propia.

con tiempos que se mantienen bajo 2 segundos para las bases de datos de *Assembly* , *BioProject y BioSample*, tiempos de respuesta muy cercanos a 0 para el genoma y un tiempo muy grande para obtener las publicaciones desde *PubMed*.

Ya que la información obtenida de cada base de datos se realiza de forma paralela, el tiempo de respuesta de la consulta completa dependerá del tiempo de respuesta mayor entre todas las consultas, ya que se necesita terminarlas todas para entregar la información al cliente.

4.4. Tiempo de procesamiento de genomas nuevos

Por último, tenemos el tiempo que demora nuestra aplicación en obtener y procesar un nuevo genoma que no se encuentra en la base de datos Mongo y, por lo tanto debe ser buscado en la base de datos de la NCBI. Se utilizaron los mismos 3 genomas de las pruebas anteriores, los cuales tenían un tamaño de 1146 *bytes*, 317918 *bytes* y 876811 *bytes*, y se obtuvieron los siguientes resultados:

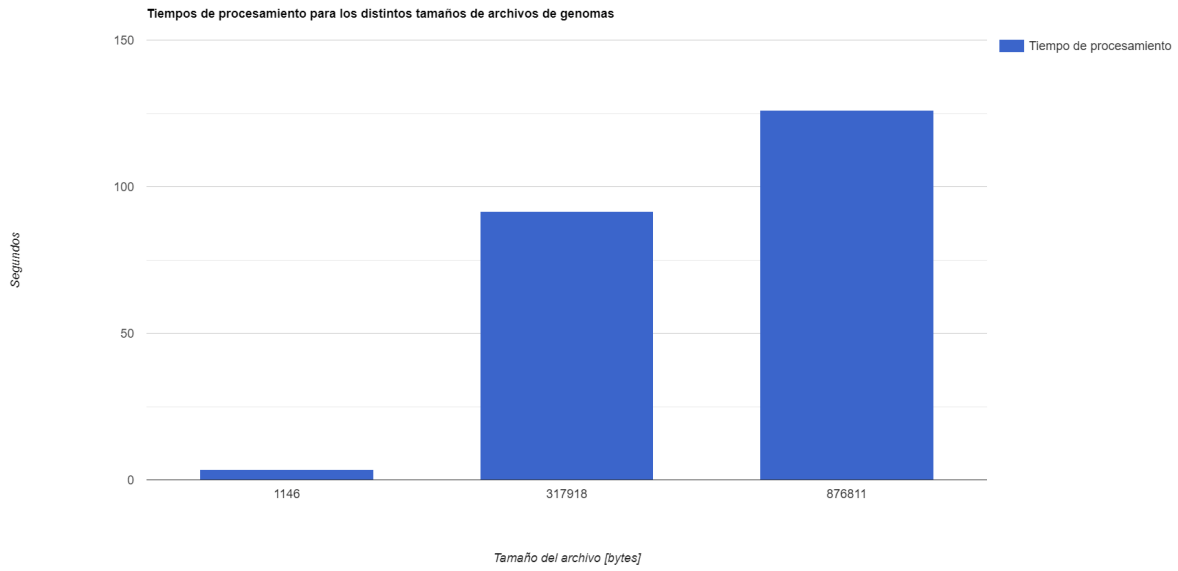

Figura 53: Tiempos de procesamiento para distintos tamaños de archivos de genomas. Fuente: Elaboración propia.

donde podemos ver que el tiempo de procesamiento, obviamente, depende completamente del tamaño de el archivo, ya que toma más tiempo la descarga y posterior obtención de datos provenientes del archivo. Podemos ver en el gráfico que el archivo de 1146 *bytes* tomó aproximadamente 3.5 segundos en ser procesado y estar listo para consultas mientras que el archivo de, 317918 *bytes* demoró aproximadamente 91.5 segundos y teniendo el mayor tiempo de procesamiento el archivo de 876811 *bytes* con aproximadamente 126 segundos.

CAPÍTULO 5
CONCLUSIONES

5.1. Objetivo general

La aplicación diseñada e implementada cumple con el objetivo principal de integrar las distintas bases de datos biológicas heterogéneas, almacenando los datos de genomas, utilizados para la visualización e identificación de contextos genómicos, en una base de datos que nos permita consultarlos sin la necesidad de descargar directamente los archivos genomas desde un servidor FTP, mientras que la información relacionada con proyectos, muestras, publicaciones, entre otros, asociados a estos genomas se obtienen desde la NCBI, entregando de forma integrada toda la información requerida mediante una única consulta.

5.2. Objetivos específicos

5.2.1. Diseño del esquema que integrará los datos

Se diseñó e implementó un esquema con el lenguaje GraphQL, que también utiliza la API construida, el cual permite obtener, almacenar y responder con los distintos tipos de datos e información sobre genomas que se necesitan para graficar los contextos de estos en conjunto con la información relacionada con los genomas provenientes de las demás bases de datos. Además, para cada base de datos integrada se creó un esquema de GraphQL que permite almacenar sus datos específicos, obtenidos mediante la API de Entrez, siendo estos esquemas utilizados por el esquema principal para mantener todo integrado en un único esquema que es el utilizado por el cliente para realizar las consultas.

5.2.2. Diseño de un mecanismo para abstraer y mapear los datos heterogéneos

Los datos de genomas son mapeados a un modelo de base de datos mediante un script de Python, modelo que esta diseñado para poder almacenar y consultar los genomas de manera más eficiente. Por parte de la API de GraphQL se utilizan los esquemas diseñados para mapear toda la información que se obtiene desde las distintas bases de datos de la NCBI y en conjunto con el modelo de la base de datos Mongo se crea una interfaz que abstrae todas las consultas realizadas y la forma en que se almacena cada dato en su respectiva base de datos para que el o los clientes puedan realizar consultas de manera rápida y fácil por medio de esta interfaz que las integra, desarrollando una herramienta robusta y que permite obtener datos de las bases de datos integradas sin la necesidad de dirigirse directamente a la NCBI para buscar los genomas o la información asociada repartida en las distintas fuentes.

5.2.3. Implementar una API utilizando el esquema diseñado

Se diseñó e implementó una API de GraphQL principalmente con el objetivo de entregar los datos necesarios para la identificación de contextos genómicos, para lo que se hace uso de los esquemas y modelo diseñados, cuyos campos corresponden a datos que necesita la interfaz encargada de graficar los contextos genómicos y que forman parte de los requisitos para esta solución en conjunto con el tipo de consultas.

5.2.4. Evaluar el rendimiento de la solución implementada

Para evaluar el rendimiento de la solución implementada se realizaron pruebas con 3 tamaños de genomas distintos, tamaño que depende de la cantidad de *features* que tenga, utilizando los distintos tipos de consultas existentes y midiendo el tiempo en que demora cada consulta en obtener los datos desde cada base de datos. Por último, se midió el tiempo en que la API demora en procesar un genoma nuevo cuando este existe en la NCBI pero no se encuentra en la base de datos Mongo y, de igual que se hizo para medir el rendimiento de las consultas, se utilizaron los mismos 3 tamaños de genomas para realizar las pruebas. Estas pruebas demostraron tiempos de respuesta pequeños que reducen el tiempo de búsqueda comparado con realizar la búsqueda manual de todos estos datos dentro de la NCBI, aportando una útil herramienta para cualquiera que necesite de estos datos biológicos.

5.3. Limitaciones

Ya que esta solución fue implementado utilizando las herramientas de Entrez, una API de la NCBI que nos permite obtener archivos desde su servidor FTP o información de las publicaciones almacenadas en sus distintas bases de datos, la principal limitación que posee es el límite de consultas que se le puede hacer a la API de Entrez, número que es indicado en las normas de uso y que corresponden a no más de 3 consultas por segundo en caso de un usuario cualquiera o 10 consultas por segundo en caso de usuarios registrados, que corresponde al caso de esta aplicación, ya que utiliza un usuario registrado en la NCBI y en caso de que se requieran más consultas por segundo o alguna operación que necesite de mucho más tiempo de ejecución, se requiere pedir autorización por correo a la NCBI.

La otra limitación existente en esta aplicación es que dependiendo del usuario y qué tipos de genomas busca utilizar, puede que estos no se encuentren en la base de datos Mongo, por lo que cada una de las consultas que realice inicialmente retornan datos nulos, ya que se tendrán que procesar y almacenar los genomas consultados, llevando a que esta API no se pueda utilizar de manera inmediata sino que se necesitan realizar consultas para poblar la base de datos.

5.4. Trabajo futuro

Con respecto al trabajo a futuro, dentro de los archivos que contienen los datos de los genomas existe información que no fue utilizada dentro de esta memoria, que no es necesaria, hasta el momento, para la visualización e identificación de contextos genómicos, pero que puede ser útil para otro tipo de trabajos, por lo que se puede seguir ampliando el esquema de la aplicación e integrar o agregar una mayor cantidad de datos, ampliando la cantidad de proyectos a los que esta API pueda ser útil. Gracias a la flexibilidad de los esquemas de graphql y las bibliotecas utilizadas para la creación de esta aplicación, si se quieren utilizar más datos simplemente se necesitan modificar los esquemas a los cuales se les quiere agregar campos y al obtener los datos desde la NCBI, almacenar los nuevos datos en los nuevos campos creados, mientras que para el caso de los genomas, el modelo de la base de datos y esquema de GraphQL es una única interfaz donde se decide qué campo pertenece al modelo y cual al esquema mediante el uso de decoradores, por lo que solo se debe agregar el nuevo campo e indicar a cual pertenece.

Los archivos de genomas contienen una gran cantidad de datos relacionados con su estructura y composición, pero además presentan una gran cantidad de referencias a autores e instituciones, pudiendo ser factible la posibilidad de realizar una API de tipo grafo que relacione todos estos datos y poder integrar no solo datos relacionados con el genoma, sino que también datos relacionados con los estudios realizados por estos autores en otros ámbitos de la biología.

La cantidad de datos biológicos es muy grande y va en constante aumento, lo que se traduce a una gran cantidad de datos que se debe manejar, entender o aprender si no eres familiar con conceptos biológicos. La NCBI presenta una extensiva documentación con todo lo necesario para entender qué significa cada palabra clave dentro de los archivos de genomas, como es el formato que debe seguir cada archivo y ciertas recomendaciones al procesar o extraer datos de estos. Además cada base de datos presenta un documento de ayuda para entender que se almacena, cómo se almacenan los datos y de donde se obtienen, siendo algunas más explicativas que otras pero sirve como introducción para entender con que se esta trabajando.

5.5. Palabras finales

El desarrollo de esta aplicación no solo utilizó conocimientos aprendidos durante los años de estudio para el modelado de bases de datos,el desarrollo y utilización de *softwares*, herramientas, entre otras cosas, sino que nuevos conocimientos que provinieron de el estudio y análisis de los datos biológicos utilizados, que se necesitaron entender para la realización de los esquemas, consultas y principalmente la integración de los datos. Además se logró integrar esta aplicación con la interfaz encargada de graficar los contextos genómicos, un trabajo que se encuentra fuera de el alcancé de esta memoria, pero que era el objetivo que se tenia en mente al crear esta API. Este trabajo me permitió entender un poco más un área interesante, y que nunca pensé volvería a retomar desde el colegio. como es la biología y si bien esta solución tiene sus limitaciones con respecto a los tiempos de respuesta y su uso inicial, presenta una buena oportunidad para trabajar con los datos que provee la NCBI abiertamente.

ANEXOS

5.1. Listado de bases de datos de la NCBI

Tabla 2: Bases de datos de la NCBI que maneja Entrez.
Fuente: Elaboración Propia.

Base de datos	Descripción	Nombre de UID
BioProject	Colección de estudios de genómica, genómica funcional y genética y enlaces a sus conjuntos de datos resultantes.	*BioProject ID*
BioSample	Base de datos que contiene descripciones de materiales biológicos usados en ensayos experimentales.	*BioSample ID*
Biosystems	Base de datos que agrupa literatura biomédica, pequeñas moleculas y datos de secuencias en términos de relaciones biológicas.	BSID
Books	Colección de libros biomédicos, la cual incluye titulos científicos, recursos genéticos como *GeneReviews*[38] y manuales de ayuda del NCBI.	Book ID
Conserved Domainss	Una colección de alineaciones de secuencias y perfiles que representan dominios proteicos conservados en la evolución molecular.	PSSM-ID
dbGaP	Centro de archivos y distribución de la descripción y los resultados de los estudios que investigan la interacción entre el genotipo y el fenotipo.	dbGaP ID

PubMed	Una base de datos de citas y resúmenes de la literatura biomédica de MEDLINE y otras revistas de ciencias de la vida.	PMID
dbVar	Desarrollada para archivar información asociada a la variación genómica a gran escala, incluyendo grandes inserciones, deleciones, translocaciones e inversiones.	dbVar ID
Genome	Contiene datos de secuencias y mapas de los genomas completos de más de 1000 organismos.	*Genome ID*
GEO Datasets	Almacena conjuntos de datos curados de expresión génica y abundancia molecular reunidos a partir del repositorio GEO.	GDS ID
GEO Profiles	Almacena perfiles individuales de expresión génica y abundancia molecular ensamblados a partir del repositorio GEO.	GEO ID
MeSH	Vocabulario controlado de la NLM para la indexación de artículos para MEDLINE/PubMed.	mesh
NCBI C++ Toolkit	Un manual completo sobre el conjunto de herramientas C++ del NCBI, que incluye su marco de diseño y desarrollo, una referencia de la biblioteca C++, ejemplos y demostraciones de software, preguntas frecuentes y notas de la versión.	*Toolkit ID*
NCBI Web Site	Página web principal de la NCBI.	*Web Site ID*
Nucleotide	Una colección de secuencias de nucleótidos de varias fuentes, incluyendo GenBank, RefSeq, la base de datos TPA y PDB.	*GI number*

NLM Catalog	Datos bibliográficos de todas las revistas, libros, audiovisuales, programas informáticos, recursos electrónicos y otros materiales que se encuentran en los archivos de la biblioteca.	*NLM Catalog ID*
PopSet	Base de datos de secuencias de ADN relacionadas que provienen de estudios comparativos: filogenéticos, poblacionales, ambientales y, en menor medida, mutacionales.	PopSet ID
Protein	Una base de datos que incluye registros de secuencias de proteínas de diversas fuentes.	*GI number*
Protein Clusters	Una colección de secuencias de proteínas relacionadas (clusters), que consiste en proteínas de secuencias de referencia codificadas por plásmidos y genomas completos de procariotas y organelos.	*Protein Cluster ID*
PubChem BioAssay	Consiste en los datos de bioactividad depositados y en las descripciones de los ensayos de bioactividad utilizados para el cribado de las sustancias químicas contenidas en la base de datos *PubChem Substance*.	AID
PubChem Compound	Contiene estructuras químicas únicas y validadas (moléculas pequeñas) que pueden buscarse utilizando nombres, sinónimos o palabras clave.	CID
PubChem Substance	Contiene información de sustancias enviada electrónicamente a PubChem por los depositantes.	SID

Gene	Una base de datos de genes en la que se pueden realizar búsquedas, centrada en los genomas que han sido completamente secuenciados y que cuentan con una comunidad de investigación activa que aporta datos específicos de los genes.	*Gene ID*
PubMed Central	Un archivo digital de literatura de revistas biomédicas y de ciencias de la vida a texto completo, incluyendo medicina clínica y salud pública.	PMCID
dbSNP	Incluye variaciones de un solo nucleótido, microsatélites e inserciones y deleciones a pequeña escala.	rs number
SRA	Almacena datos de secuenciación de la siguiente generación de plataformas de secuenciación, incluyendo *Roche 454 GS System®*, *Illumina Genome Analyzer®*, *Life Technologies AB SOLiD System®*, *Helicos Biosciences Heliscope®*, *Complete Genomics®* y *Pacific Biosciences SMRT®*.	SRA ID
Structure	Contiene estructuras macromoleculares en 3D derivadas del Banco de Datos de Proteínas, así como herramientas para su visualización y análisis comparativo.	MMDB-ID
Taxonomy	Contiene los nombres y linajes filogenéticos de más de 160.000 organismos que tienen datos moleculares en las bases de datos del NCBI.	TaxID

REFERENCIAS BIBLIOGRÁFICAS

[Bechhofer *et al.*, 2009] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., y Stein, L. A. (2009). Owl: Web ontology language.

[Berners-Lee *et al.*, 2001] Berners-Lee, T., Hendler, J., y Lassila, O. (2001). The semantic web. *Scientific american*, 284(5):34–43.

[Bichutskiy *et al.*, 2006] Bichutskiy, V. Y., Colman, R., Brachmann, R. K., y Lathrop, R. H. (2006). Heterogeneous biomedical database integration using a hybrid strategy: A p53 cancer research database. *Cancer informatics*, 2:117693510600200021.

[Box *et al.*, 2000] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., y Winer, D. (2000). Simple object access protocol (soap) 1.1.

[Buitelaar *et al.*, 2008] Buitelaar, P., Cimiano, P., Frank, A., Hartung, M., y Racioppa, S. (2008). Ontology-based information extraction and integration from heterogeneous data sources. *International Journal of Human-Computer Studies*, 66(11):759–788.

[Buron *et al.*, 2019] Buron, M., Goasdoué, F., Manolescu, I., y Mugnier, M.-L. (2019). Ontology-based rdf integration of heterogeneous data.

[Christensen *et al.*, 2001] Christensen, Erik and Curbera, Francisco and Meredith, Greg and Weerawarana, Sanjiva and others (2001). Web services description language (wsdl) 1.1.

[Fielding *et al.*, 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., y Berners-Lee, T. (1999). Hypertext transfer protocol–http/1.1.

[Fielding y Taylor, 2000] Fielding, R. T. y Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures*, volumen 7. University of California, Irvine Irvine.

[Gagnon, 2007] Gagnon, M. (2007). Ontology-based integration of data sources. En *2007 10th International Conference on Information Fusion*, pp. 1–8. IEEE.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220.

[Khoumbati *et al.*, 2005] Khoumbati, K., Themistocleous, M., e Irani, Z. (2005). Integration technology adoption in healthcare organisations: A case for enterprise application integration. En *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pp. 149a–149a. Ieee.

[Klyne y Carroll,] Klyne, G. y Carroll, J. Resource description framework (rdf): Concepts and abstract syntax (feb 2006).

[Levy, 1999] Levy, A. Y. (1999). Combining artificial intelligence and databases for data integration. En *Artificial Intelligence Today*, pp. 249–268. Springer.

[Levy y Minker, 2000] Levy, A. Y. y Minker, J. (2000). Logic-based techniques in data integration, logic-based artificial intelligence.

[Messaoudi *et al.*, 2020] Messaoudi, C., Fissoune, R., y Badir, H. (2020). Ipds: A semantic mediator-based system using spark for the integration of heterogeneous proteomics data sources. *Concurrency and Computation: Practice and Experience*, p. e5814.

[Ramesh y Ram, 1997] Ramesh, V. y Ram, S. (1997). Integrity constraint integration in heterogeneous databases: An enhanced methodology for schema integration. *Information Systems*, 22(8):423–446.

[Stünkel *et al.*, 2020] Stünkel, P., von Bargen, O., Rutle, A., y Lamo, Y. (2020). Graphql federation: A model-based approach. *Journal of Object Technology*, 19(2).

[Tang y Fan, 2016] Tang, E. y Fan, Y. (2016). Performance comparison between five nosql databases. En *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pp. 105–109. IEEE.

[Ulrich *et al.*, 2019] Ulrich, H., Kern, J., Tas, D., Kock-Schoppenhauer, A.-K., Ückert, F., Ingenerf, J., y Lablans, M. (2019). Ql 4 mdr: a graphql query language for iso 11179-based metadata repositories. *BMC medical informatics and decision making*, 19(1):1–7.

[Wiederhold, 1992] Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE computer*, 25(3):38–49.

[Yoon *et al.*, 2017] Yoon, B.-H., Kim, S.-K., y Kim, S.-Y. (2017). Use of graph database for the integration of heterogeneous biological data. *Genomics & informatics*, 15(1):19.