

# **Trabajo de Fin de Grado de la titulación del Grado en Ingeniería Informática: Intensificación en Tecnologías de la Información (ULPGC)**

Interfaz web para el acceso a datos genéticos de pacientes de  
Gran Canaria

**Autor**

Alberto Brandon Velázquez

**Curso Académico**

2021/2022

**Tutores**

José Javier Lorenzo Navarro

Pascual Lorente Arencibia

Antonio Tugores Cester

# Resumen

Dos de los grandes objetivos de las ciencias de la salud son el diagnóstico de enfermedades en fases temprana o incluso antes de que lleguen a manifestarse y tratarlas con eficiencia o, incluso, prevenirlas. Para cumplir estas metas, es necesario conocer no solo los síntomas, sino las causas de cada patología. Una de estas causas son las variaciones genéticas, que es lo que hace que todos los individuos de la misma especie sean diferentes.

El objetivo de este proyecto es facilitar el análisis de estas variaciones genéticas para poder determinar cuál es la causa de una enfermedad. En concreto, este trabajo pretende añadir lógica y funcionalidades a un conjunto de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones, conectada a una base de datos con los datos genéticos de pacientes, para permitir visualizar, filtrar y buscar qué variantes son compartidas por uno o más pacientes y cuáles de ellas podrían ser la causa de enfermedad.

# Abstract

Two of the great objectives of the health sciences are diagnosis of diseases in early stages or even before they manifest themselves and treating them efficiently or even preventing them. To meet these goals, it is necessary to know not only the symptoms, but also the causes of each pathology. One of these causes is genetic variation, which is what makes all individuals of the same species different.

The objective of this project is to facilitate the analysis of these genetic variations in order to determine what is the cause of a disease. Specifically, this work aims to add logic and functionalities to a set of definitions and protocols that are used to design and integrate the application software, connected to a database with the genetic data of patients to allow visualization, filter and find which variants are shared by one or more patients and which of them could be the cause of disease.

# Agradecimientos

Agradecer a mis tutores Pascual, Paqui, Antonio y José Javier Lorenzo Navarro. Tanto por tener paciencia a lo largo de este proyecto del cual había mucho contenido desconocido para mí. Como por ayudarme, guiarme y animarme en este último tramo de mi carrera.

Agradecer a mis padres, hermano, pareja y familia por estar conmigo y darme los ánimos necesarios para llevar a cabo todo lo que me he propuesto desde mi comienzo en el mundo universitario.

# ÍNDICE GENERAL

1.	Introducción .....	11
1.1	Motivación .....	11
1.2	Objetivos .....	11
1.3	Competencias específicas del proyecto .....	12
1.4	Normativa y legislación .....	13
2.	Estado del arte .....	14
3.	Herramientas .....	18
3.1	Tecnologías aplicadas .....	18
3.1.1	Java .....	18
3.1.2	Spring .....	18
3.1.3	PostgreSQL .....	19
3.2	Software empleado en el desarrollo .....	19
3.2.1	IntelliJ .....	19
3.2.2	GIT .....	20
3.2.3	Dbeaver .....	20
3.2.4	Postman .....	21
3.2.5	Hardware .....	21
3.3	Capas del backend del aplicativo .....	22
3.3.1	Base de datos .....	22
3.3.2	Modelo .....	22
3.3.3	Repositorios .....	22
3.3.4	Controladores .....	23
4.	Análisis y desarrollo .....	24
4.1	Definiciones de Interés .....	24

4.1.1	Características .....	25
4.1.2	Diferencias entre el uso del GET y el POST .....	26
4.1.3	Ventajas del método GET .....	27
4.1.4	Ventajas del método POST .....	27
4.2	Capas del proyecto .....	27
4.3	Desarrollo.....	29
4.4	Fases del proyecto.....	29
4.5	Tipos de resultados .....	43
4.5.1	Objetos .....	43
4.5.2	Lista.....	44
4.5.3	Páginas .....	44
4.6	Genes.....	45
4.7	BioTypes .....	48
4.8	Chromosomes .....	49
4.9	Variants .....	50
4.10	Populations.....	50
4.11	Individuals.....	51
4.11.1	Impacts .....	52
4.11.2	Effects .....	52
5.	Conclusión .....	54
5.1	Trabajo futuro .....	55
6.	Bibliografía .....	56

# Índice de imágenes

Imagen 1. Ácido desoxirribonucleico .....	15
Imagen 2. Esquema del proceso de traducción genética sintetizada por una proteína a partir del ARN mensajero en un ribosoma .....	16
Imagen 3. Icono de Java.....	18
Imagen 4. Icono de Spring .....	19
Imagen 5. Icono de PostgreSQL .....	19
Imagen 6. Icono de IntelliJ.....	20
Imagen 7. Icono de git .....	20
Imagen 8. Icono de DBeaver .....	20
Imagen 9. Icono de POSTMAN.....	21
Imagen 10. Esquema de las capas del backend de la REST API.....	22
Imagen 11. Esquema de la Base de Datos .....	28
Imagen 12. Esquema de la estructura del proyecto.....	28
Imagen 13. Estructura del proyecto .....	30
Imagen 14. Clase de error generada.....	31
Imagen 15. Clase para la obtención de datos por POST de genes .....	31
Imagen 16. Clase para la obtención de datos por POST de variantes.....	32
Imagen 17. Clase de genotipos .....	32
Imagen 18. Modelo de genes .....	33
Imagen 19. Repositorio de genes. ....	34
Imagen 20. Repositorio de effects .....	34
Imagen 21. Controladores de genes .....	35
Imagen 22. Implementación del repositorio de variantes .....	36
Imagen 23. Implementación del repositorio de variantes. ....	36
<i>Imagen 24. Proyección de la clase de variantes.....</i>	<i>37</i>
Imagen 25. Clase para la recogida de los parámetros enviados por POST.....	38
Imagen 26. Getters de la clase que recoge los parámetros enviados por POST .....	39
Imagen 27. Controlador de variantes .....	40
Imagen 28. Uso de EntityGraph en la proyección de variantes .....	40
Imagen 29. Esquema de la realización de test del proyecto .....	40

Imagen 30. Test del proyecto.....	41
Imagen 31. Ejemplo de JSON.....	41
Imagen 32. Test que se realizan .....	42
Imagen 33. Ejemplo de ejecución de test.....	42
Imagen 34. Modelo de datos de la tabla de genes.....	43
Imagen 35. Consulta del GET.....	44
Imagen 36. Listado de genes.....	44
Imagen 37. Campos de paginación .....	45
Imagen 38. Comparación de tiempos de ejecución de variantes con 2 parámetros en el inicio del proyecto.....	54



# Índice de tablas

Tabla 1. Parámetros que pasan a genes/ {id} .....	46
Tabla 2. Parámetros de genes o que se pasan a genes .....	46
Tabla 3. Campos de la clase de genes .....	46
Tabla 4. Campos de la clase Biotypes .....	48
Tabla 5. Campos de la clase chromosomes .....	49
Tabla 6. Campos de la clase Variants .....	50
Tabla 7. Campos de la clase chromosomes .....	51
Tabla 8. Campos de la clase individuals .....	51
Tabla 9. Campos de la clase impact .....	52
Tabla 10. Campos de la clase effect .....	53

# Índice de cuadros

Cuadro 1. Ejemplo uso del repositorio .....	23
Cuadro 2. Resultados de la consulta de genes .....	47
Cuadro 3. Resultado de la consulta de genes/{id} .....	47
Cuadro 4. Resultado de la consulta de Biotypes.....	48
Cuadro 5. Consulta de chromosomes.....	49
Cuadro 6. Consulta de populations (id, code, name) .....	51
Cuadro 7. Consulta de populations (id, code, name) .....	51
Cuadro 8. Consulta de individuals .....	51
Cuadro 9. Consulta de impact.....	52
Cuadro 10. Consulta de Effect .....	53
Cuadro 11. Consulta de Effect .....	53

# Capítulo 1

## Introducción

### 1.1 Motivación

Cada vez son más los datos que genera la secuenciación masiva de genoma, en concreto la lista de variantes genéticas específicas de cada paciente y sus propiedades. Por ello, su transmisión, almacenamiento y análisis se debe hacer utilizando las tecnologías más apropiadas. Es importante destacar la búsqueda de patrones genéticos para poder identificar y determinar enfermedades que se pueden encontrar en el paciente gracias a la molécula llamada *ácido desoxirribonucleico* (en adelante ADN).

En el caso del CHUIMI, se ha decidido convertir los clásicos archivos de texto plano en una base de datos. Para consultar estos datos, se está desarrollando tanto una interfaz web como una aplicación web, que desacopladas, se pueden desarrollar por separado. El ámbito de este trabajo se centrará en aplicar los conocimientos aprendidos durante la formación del alumno para mejorar la interfaz web, aumentando la robustez y añadiendo los puntos de entrada restantes.

### 1.2 Objetivos

Los objetivos planteados inicialmente dentro de la realización de este Trabajo de Fin de Título son los siguientes:

- Obtener la lista de variantes con todas sus propiedades de manera paginada.
- Filtrar la lista por: cromosoma, posición, gen, impacto, frecuencia en la población o genotipo por individuo.
- Descargar la lista en formato JSON, CSV o VCF.
- Obtener la lista de cromosomas

- Obtener la lista de genes
- Obtener la lista de impactos

Con la consecución de los anteriores objetivos, se proporciona una solución respecto al objetivo principal, que es proveer al CHUIMI de una infraestructura digital sobre la cual puedan desarrollar una actividad científica de una manera rápida y eficiente, dado que esto puede ayudar a salvar vidas.

## 1.3 Competencias específicas del proyecto

Las competencias cubiertas durante realización de este proyecto son las siguientes:

*“IS01 - Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software (EII,2013).”*

Debido a que una de las partes principales del proyecto fue la implementación de nuevas funcionalidades dentro de una aplicación existente, la capacidad para desarrollar servicios que cumplieran con las necesidades del cliente, en este caso el hospital, fueron necesarias para el éxito de este proyecto. A medida que se iban generando nuevas secciones en el proyecto fue necesario asegurar su mantenibilidad y funcionamiento, así como su capacidad de satisfacer al usuario.

*“SI02 - Capacidad para determinar los requisitos de los sistemas de información y comunicación de una organización atendiendo a aspectos de seguridad y cumplimiento de la normativa y la legislación vigente (EII,2013).”*

Cuando se tuvo que decidir el uso del GET y el POST era necesario asegurar de qué manera iban a ir los datos para no tener brechas de seguridad. También cuando se realizaban algunas consultas de paginación había que determinar bien el tamaño para que cuando se realizan demasiadas consultas, no saturar el servidor, de esta manera también se previenen los ataques DDoS.

También ciertos endpoints no son accesibles a no ser que el usuario introduzca un token de autorización en la cabecera HTTP "Authorization".

*“IS03 - Capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles (EII,2013).”*

Esta fue la competencia clave cuando se tocó el proyecto por primera vez, aplicando estrategias de reingeniería para poder cumplir los objetivos de este trabajo. Además de haber jugado un papel importante durante el desarrollo de algunas de las nuevas funcionalidades.

## 1.4 Normativa y legislación

Este proyecto está sujeto a los estatutos expuestos por la Ley Orgánica 3/2018, de 5 de diciembre, de protección de Datos Personales y garantía de los derechos digitales, que tiene como objetivo proteger los derechos de las personas al salvaguardar sus datos personales.

Esto se debe a que la información de cada variante y, por extensión, la de cada muestra con la que está relacionada, es información sensible que pertenece a varios pacientes a los cuales se les ha secuenciado el genoma.

Es por esto por lo que la aplicación garantiza la protección de estos datos sensibles en todo momento en las respuestas que ofrece la REST API con sus servicios.

Dado que la base de datos que contiene la información de las variantes de cada paciente está anonimizado, no es posible determinar para una muestra concreta a qué persona le pertenece esa muestra debido a que a cada muestra se le asigna un pseudónimo con el que no es posible asociarlo con un individuo concreto.

Por ejemplo, Juan Pérez tiene el pseudónimo S\_037, la BBDD sólo almacena el pseudónimo S\_037 y, por tanto, no es posible averiguar que realmente esa muestra pertenece a Juan Pérez.

Desde el punto de vista de la aplicación, lo que realmente está consiguiendo es restringir el acceso a los datos anonimizados de esa BBDD, impidiendo que una persona no autorizada pueda acceder a los datos.

## Capítulo 2

### Estado del arte

El ADN es la molécula que contiene las instrucciones biológicas que determina la información que necesita un ser vivo para su funcionamiento y desarrollo, y que hace que cada especie y cada individuo dentro de ella sean únicos. El ADN pasa de los organismos adultos a sus descendientes durante la reproducción de forma que la progenie hereda la mitad del ADN del padre y la otra mitad de la madre, que además aporta el ADN mitocondrial. El ADN aporta las instrucciones necesarias para que el ser vivo pueda desarrollarse, sobrevivir y reproducirse.

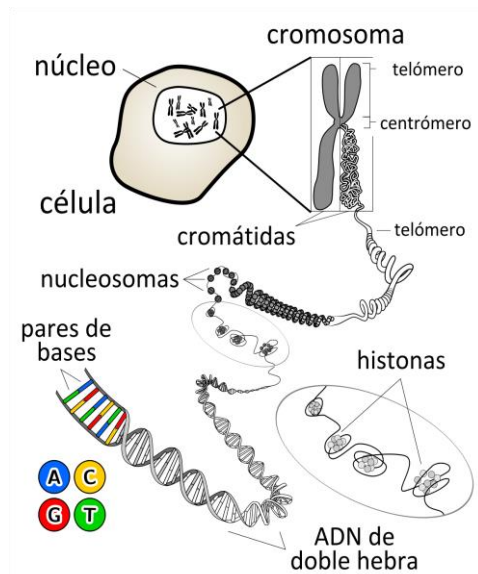


Imagen 1. Ácido desoxirribonucleico. Fuente: Wikipedia (2022)

En el organismo, el ADN se encuentra en el núcleo de las células de forma muy compacta y precisa como cromosomas. Durante la replicación el ADN se desenrolla para que pueda ser copiado o para transcribir instrucciones se utilicen en la fabricación de proteínas y procesos biológicos.

El ADN es un polímero formado por nucleótidos, unos componentes químicos que incluyen un grupo de fosfato, azúcar y una de cuatro tipos de bases nitrogenadas, formando cadenas para formar una hebra del ADN. Los cuatro tipos de bases nitrogenadas encontradas en los nucleótidos son: adenina (A), timina (T), guanina (G) y citosina (C).

En nuestra especie, las células poseen un total de 46 cadenas de ADN, en el que su genoma consta de 3 mil millones de bases organizados en 23 pares de cromosomas y conteniendo alrededor de 20.000 genes cuya información, en su mayor parte, están fragmentadas en exones, separados entre sí por intrones.

Asimismo, para realizar la mayor parte de las funciones, las cadenas de ADN deben ser transcritas a mensajes que se traduzcan para la fabricación de proteínas. Dichas secuencias, conocidas como codificantes, suponen menos del 2% del genoma en humanos. Para ello, la ARN polimerasa lee la información en una molécula del ADN y la transcribe a una molécula intermedia conocida como ácido ribonucleico mensajero o ARNm. El contenido de la información se lee en tripletes de nucleótidos y se traduce en los ribosomas a una secuencia de aminoácidos, cada uno de los 20 que existen, colocados en diversos órdenes para formar varias proteínas.

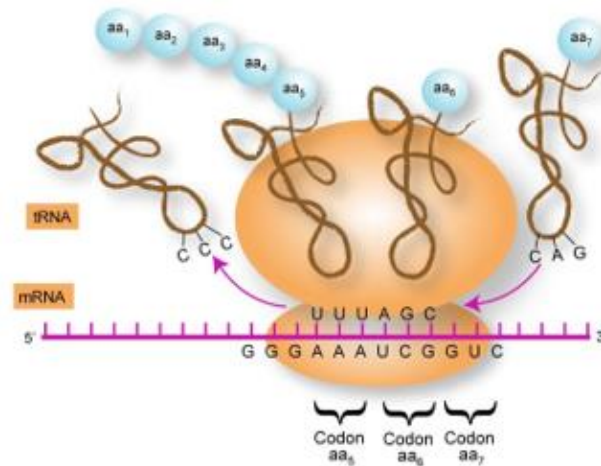


Imagen 2. Esquema del proceso de traducción genética sintetizada por una proteína a partir del ARN mensajero en un ribosoma. Fuente: Wikipedia (2022)

Muchos de los cambios genéticos o mutaciones vistas hasta hoy en día afectan a este proceso, y son el origen de enfermedades hereditarias o familiares, como la enfermedad de Wilson o la betatalasemia, las diabetes o la artritis reumatoide (OMIM, 2022; NIH, 2022).

En la unidad de investigación de CHUIMI, se usa la secuenciación de genoma completo para la detección de mutaciones en los casos de enfermedades de etiología desconocida. Todas las variantes genéticas son tenidas en cuenta por *Combination and Annotation Tool* o COAT (Henríquez-González, J., 2017) que combina casos y controles y anota las variantes en *Variant Effect Predictor* o VEP (McLaren W. et al., 2016). La *Variant Call Format* o VCF (*The Variant Call Format*, 2022), interfaz gráfica, es la que permite aplicar filtros en base a las anotaciones como: frecuencias de poblaciones (gonmAD browser, 2020), variantes intergénicas, variantes sin riesgo de acuerdo con algoritmos de predicción como SFT o *Polyphen*, entre otros.

Para terminar, la aplicación Poirot integrada en el CHUIMI, es la que prioriza las variantes que se encuentran de los tejidos en los que se expresan y, a su vez, la relación con su patología. Las dos últimas aplicaciones ya mencionadas, son herramientas que se han ido creando con CHUIMI, buscando en todo momento el desarrollo para adaptar su funcionalidad a los usuarios que necesiten analizar la secuencia de genomas para la detección de variantes patológicas. En resumen, su finalidad es intentar generar una interfaz que de sencillez a los usuarios que lo usen.





# Capítulo 3

## Herramientas

A continuación, se hablará de los diversos recursos tecnológicos usados para desarrollar este proyecto:

### 3.1 Tecnologías aplicadas

#### 3.1.1 Java

Java es una tecnología que se usa para el desarrollo de aplicaciones que convierten a la Web en un elemento más interesante y útil. Java no es lo mismo que *javascript*, se trata de una tecnología sencilla que se usa para crear páginas web y solamente se ejecuta en el explorador.

Hemos trabajado con él debido a las múltiples ventajas que nos ofrece que son: es gratuito, es un lenguaje orientado a objetos, posee una gran cantidad de librerías y Java ofrece APIs para I/O, XML parsers, conexiones de bases de datos, etc.



*Imagen 3. Icono de Java. Fuente: (Krill, 2022)*

#### 3.1.2 Spring

Spring es un *framework* de código abierto para la creación de aplicaciones empresariales Java, con soporte para Groovy y Kotlin. Tiene una estructura modular y una gran flexibilidad para implementar diferentes tipos de arquitectura según las necesidades de la aplicación, también permite pruebas unitarias, de integración y tiene una arquitectura Modelo Vista Controlador (*MVC*).



*Imagen 4. Icono de Spring. Fuente: (Spring, 2022)*

### 3.1.3 PostgreSQL

Es un gestor que trabaja con bases de datos relacionales. Hemos escogido este gestor por las ventajas que tiene, como, por ejemplo: Es de código abierto, cumple con el modelo ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), ofrece opciones que otras bases de datos no tienen (como la posibilidad de crear tablas heredadas, esquemas o triggers), etc.



*Imagen 5. Icono de PostgreSQL. Fuente: (PostgreSQL, 2022)*

## 3.2 Software empleado en el desarrollo

### 3.2.1 IntelliJ

IntelliJ IDEA es un entorno de programación inteligente y sensible al contexto para trabajar con Java y otros lenguajes JVM.

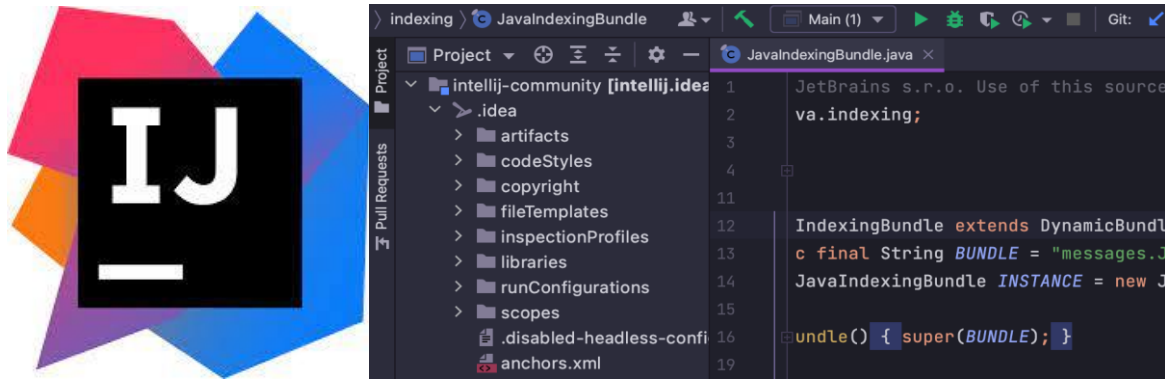


Imagen 6. Icono de IntelliJ. Fuente: (IntelliJ IDEA, 2022)

### 3.2.2 GIT

Tecnología dedicada al control de versiones, la cual facilita mantener un flujo de trabajo controlado durante toda la vida de un proyecto.



Imagen 7. Icono de git. Fuente: (git, 2022)

### 3.2.3 Dbeaver

DBeaver es un software que actúa como una herramienta de base de datos universal destinada a desarrolladores y administradores de bases de datos.

Soporta todas las bases de datos más populares como: MySQL, PostgreSQL, MariaDB, SQLite, Oracle, DB2, SQL Server, Sybase, MS Access, Teradata, Firebird, Derby, etc.

En nuestro caso lo hemos usado para ver las tablas de datos y sobre todo como entorno de prueba para generar sql que se tienen que generar en nuestra API.



Imagen 8. Icono de DBeaver. Fuente: (Dbeaver Community, 2022)

### 3.2.4 Postman

Postman es una herramienta que se utiliza, sobre todo, para el testing de API REST, aunque también admite otras funcionalidades como por ejemplo, trabajar con entornos (calidad, desarrollo, producción) y de este modo es posible compartir a través de un entorno cloud la información con el resto del equipo involucrado en el desarrollo.

En nuestro caso lo hemos usado para automatizar la realización de consultas, realizando 1000 iteraciones con un retardo de 0ms. Esto nos ayuda a ver si los tiempos de respuesta están en lo estimado y si hay riesgo de denegación de servicios.



*Imagen 9. Icono de POSTMAN. Fuente: (SNGULAR, 2022)*

### 3.2.5 Hardware

El recurso de hardware utilizado en este proyecto será descrito a continuación con sus características.

#### **Equipo personal (Portátil):**

- **Nombre del SO:** Microsoft Windows 11 Home
- **Procesador:** Intel(R) Core(TM) i7-10750H CPU@2.60GHz, 2592Mhz, 6 procesadores principales, 12 procesadores lógicos
- **Memoria física instalada (RAM):** 16 GB

Durante el proyecto se ha utilizado el equipo personal dado que todo el desarrollo del mismo se ha realizado de manera local. Aunque el UICHUIMI tiene un servidor con las siguientes características:

- Intel(R) Core™ i9-9980XE CPU @ 3.00 GHz – 18 Cores (36 hilos).

- 1200-4500 Mhz.
- 32 GB RAM DDR4 (4x8).

En este servidor donde se alojan y ejecutan todos los componentes del proyecto:

- La REST API con los *endpoints* necesarios para acceder a los servicios.
- La base de datos PostgreSQL con toda la información de las variantes genéticas.
- El proyecto Angular desplegado para dar acceso a la aplicación web.

### 3.3 Capas del backend del aplicativo



*Imagen 10. Esquema de las capas del backend de la REST API. Fuente: Excalidraw (2022)*

#### 3.3.1 Base de datos

Una base de datos es una recopilación organizada de información o datos estructurados, que normalmente se almacena de forma electrónica en un sistema informático. Esta capa es la primera de todas dado que los datos son necesarios para poder dar toda la información necesaria a las consultas de los clientes del aplicativo.

#### 3.3.2 Modelo

En java, el modelo se define, por regla general, como una clase asociada a cada tabla de la base de datos. Cada objeto de estas clases representa, por lo tanto, una fila en la base de datos.

#### 3.3.3 Repositorios

Los repositorios son instancias en Java que permiten leer y escribir en la base de datos utilizando instancias de las clases definidas en el modelo. Los repositorios hacen transparente al desarrollador, las sentencias SQL.

El módulo JPA de Spring Data contiene un espacio de nombres personalizado que permite definir beans de repositorio. También contiene ciertas características y atributos de elementos que son especiales para JPA. Generalmente, los repositorios JPA se pueden configurar mediante el elemento, como se muestra en el siguiente ejemplo: `repositories`.

```
<jpa:repositories base-package="com.acme.repositories" />
```

*Cuadro 1. Ejemplo uso del repositorio. Fuente: elaboración propia*

### 3.3.4 Controladores

Este componente es el que responde a la interacción que hace el usuario en la aplicación y realiza las peticiones al modelo para pasar estos a la vista y este sirve de puente entre la petición y la base de datos con la conexión de los repositorios.

# Capítulo 4

## Análisis y desarrollo

### 4.1 Definiciones de Interés

#### REST API o API REST

Una Rest API es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful, basado en la manipulación de recursos.

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones. Las API permiten que los productos y servicios se comuniquen entre sí o con otros, sin necesidad de saber cómo están implementados.

En otras palabras, las API te permiten interactuar con un ordenador o sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla.

El uso de la API en la unidad de investigación del Complejo Hospitalario Universitario Insular Materno infantil (en adelante CHUIMI), nos proporciona múltiples beneficios a la hora de mejorar los procesos.

- Separación entre el cliente y el servidor, gracias a esto, se mejora la portabilidad de la interfaz a otro tipo de plataformas, también, aumenta la escalabilidad de los proyectos y permite que los distintos componentes desarrollados puedan evolucionar de forma independiente.
- Hay visibilidad, fiabilidad y escalabilidad. Es evidente que la separación entre clientes y servidor permite que cualquier equipo de desarrollo pueda escalar el producto sin excesivos problemas. Por tanto, permite que se pueda migrar a otros servidores o



realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta.

- La API REST siempre es independiente del tipo de plataformas o lenguajes, por tanto, se adapta al tipo de plataformas con las que se esté trabajando. Esto ofrece una gran libertad a la hora de probar o cambiar nuevos entornos.
- La API REST permite tener servidores PHP, Java, Python o Node.js, no obstante, se debe tener en cuenta que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usando XML o JSON.

En definitiva, las REST API nos permiten introducir nuevas funcionalidades en nuestro sitio web que pertenezcan a otro, de esta manera, podemos añadir valor a nuestros productos o servicios.

### 4.1.1 Características

A continuación, describiré las características de las API REST y qué las convierte en una herramienta tan útil.

#### 1. Protocolo cliente/servidor sin estado

Cada petición HTTP contiene toda la información necesaria para ejecutarla, por tanto, esto permite que ni el cliente ni el servidor necesiten recordar ningún estado previo. No obstante, existen algunas excepciones y hay algunas aplicaciones HTTP que incorporan memoria caché como la nuestra, para que así, el cliente pueda ejecutar en un futuro la misma respuesta para peticiones idénticas.

#### 2. Cuatro operaciones más importantes

Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro:

- POST (crear)
- GET (leer y consultar)
- PUT (editar)
- DELETE (borrar)

En nuestro caso solo son necesarias POST y GET dado que no se va a modificar ni eliminar ningún dato.

### **3. Objetos en REST manipulados con URI**

La URI es un identificador único de cada recurso de un sistema REST. Esto nos facilita el acceso a la información, para poder modificarla o borrarla. También para compartir su ubicación exacta a terceros.

### **4. Interfaz uniforme**

Para poder realizar una transferencia de datos en una API, se aplican acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto permite facilitar la existencia de una interfaz uniforme que sistematiza el proceso con la información.

### **5. Sistema de capas**

Su estructura es jerárquica entre sus componentes, y cada una de estas capas, lleva a cabo una funcionalidad dentro del sistema REST.

### **6. Utilización de Hipermedios**

El concepto hipermedio es utilizado en los casos de API REST dado que sirve para explicar la capacidad de una interfaz de desarrollo de aplicaciones para proporcionar al cliente y a los usuarios los enlaces adecuados, y ejecutar acciones concretas sobre los datos.

Debemos tener en cuenta que cualquier API debe disponer de hipermedios, puesto que este principio es el que define que cada vez que se hace una petición al servidor este de una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente.

## **4.1.2 Diferencias entre el uso del GET y el POST**

De inicio en el proyecto se tuvo que debatir sobre qué era lo mejor para usar en cada endpoint. Principalmente se iba a utilizar solo GET, pero viendo algunas de las ventajas del POST decidimos mezclar ambas y coger el POST para algunas cosas que nos interesan como puede ser la de generar URLs más simples.

### 4.1.3 Ventajas del método GET

Asimismo, las ventajas del método GET son las siguientes:

- Codificar el código HTML necesario para enviar un dato mediante una petición get (un enlace que adjunta variables) es más sencillo, no necesitamos crear ningún formulario, simplemente creamos enlaces comunes, que envíen los datos necesarios al ser pulsados por el usuario.
- Las peticiones GET se pueden almacenar en caché.

### 4.1.4 Ventajas del método POST

Sin embargo, las ventajas del método POST son:

- Permite un tamaño de carga útil de 2048 bytes.
- Permite más flexibilidad a la hora de pasar parámetros.
- Es uno de los métodos más seguro para rellenar formularios con datos sensibles.
- Te permite añadir parámetros o realizar búsquedas jerárquicas.

## 4.2 Capas del proyecto

A continuación, se muestra la estructura por capas del proyecto compuestas por: Modelo de datos, estructura y testeo. Se empezará hablando del modelo que diseña el proyecto, viendo cómo se relacionan dichas tablas.

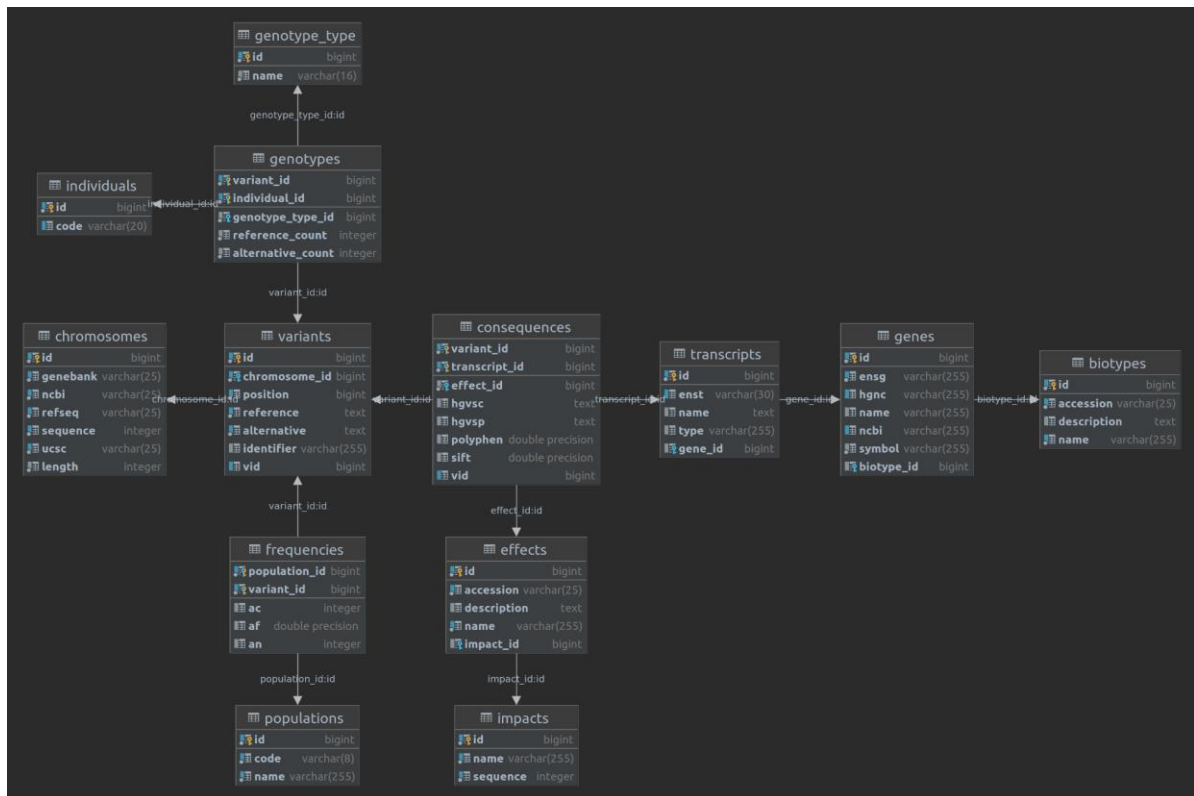


Imagen 11. Esquema de la Base de Datos. Fuente: (PostgreSQL, 2022) elaboración del UICHUIMI

**Modelo de Datos** (como vemos en la anterior imagen 10): La tabla de variantes es el centro de todo el modelo, datos de localización (*chromosomes*, *position*, *transcript*, *genes*, *biotype*), datos de población (*frequencies*, *population*, *genotypes*), datos de impacto (*consequence*, *effect*, *impact*)



Imagen 12. Esquema de la estructura del proyecto. Fuente: elaboración propia, app Excalidraw (2022)

### Estructura del Proyecto (Imagen 12):

A pesar de que la estructura de todo el proyecto (*Back-End*) fue la misma durante todo este desarrollo, el modelo de datos fue evolucionando a medida que se iban implementando nuevas funcionalidades a la REST API. Con el fin de facilitar el manejo de los datos y proveer más información clara y concisa a los usuarios.

Toda la aplicación se encuentra alojada dentro de un servidor a la cual se accede mediante protocolo HTTP. Al acceder a la dirección de la aplicación Angular, esta es la que

realiza las peticiones contra la RestAPI dentro del mismo servidor. Luego es la RestAPI la que recupera los datos accediendo a la base de datos en PostgreSQL. La respuesta una vez recuperada es mandada a la aplicación Angular en formato JSON, donde finalmente Angular procesa los datos para quedarse únicamente con la información que necesita, así como prepararla para el usuario, y es entonces mostrada en la interfaz de usuario.

## 4.3 Desarrollo

El desarrollo de los objetivos se ha realizado por orden de sencillez, para permitir una mejor adaptación y conocimiento tanto del proyecto como de algunas herramientas que eran nuevas para mí, antes de abordar el objetivo principal.

En primer lugar, empezamos a generar el modelo de datos, repositorios y controladores de cada objeto. Teniendo en cuenta en los controladores la forma de devolución de la información (Objetos, listas y páginas).

En segundo lugar, se desarrolló una implementación en el repositorio de variantes para poder realizar las consultas correspondientes a la base de datos y obtener la información necesaria según el parámetro que nos pasaran por POST en la tabla de variantes.

En tercer lugar, se realizaron modificaciones para que en la devolución del JSON solo aparecieran los campos determinados que les interesaban a los clientes, de esta manera se quedaba un documento algo más sencillo con la información necesaria.

En cuarto lugar, se modificó el controlador de variantes, para poder recoger una serie de filtros que el cliente pasaría por POST y de reducir el número de consultas SQL.

Finalmente, se generaron una serie de test para ver si la app estaba funcionando correctamente y ver cómo iban cambiando los tiempos de ejecución durante el desarrollo.

## 4.4 Fases del proyecto

A continuación, se explican las fases en las que se dividió la realización de este proyecto:

### **Primera fase – Modelo de datos**

Como se comentó en el punto 5 ya existía un proyecto funcional implementado en Java para el *Back-End* de la aplicación web con el que hubo que familiarizarse primero. Esto se logró mediante la refactorización del proyecto y del código en dos pasos.

## 1.- Reestructuración del proyecto

El primer paso de esta primera fase consistió en realizar una reestructuración del código y cómo se distribuían los componentes en el proyecto, con el objetivo de facilitar su mantenimiento y lectura. Para ello se realizó lo siguiente, se crearon distintas secciones para almacenar modelos, repositorios y controladores durante todo el proyecto.

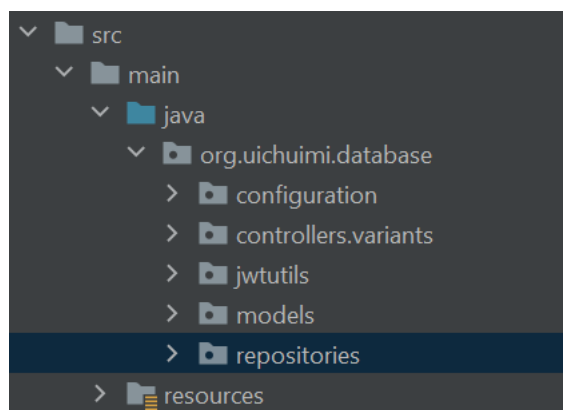


Imagen 13. Estructura del proyecto. Fuente: elaboración propia, app IntelliJ IDEA (2022)

Siguiendo esta jerarquía, cada fichero generado se almacena en su sitio correspondiente para dejar las cosas de manera clara y facilitando el entendimiento a un futuro desarrollador.

## 2.- Refactorización de código

Aprovechando el cambio estructural hecho en el paso anterior se fueron aplicando gradualmente las siguientes mejoras:

- Eliminar importaciones de librerías innecesarias.
- Generar métodos de errores para que puedan ser utilizados en futuras mejoras y de esta manera evitar código espagueti, como vemos en la imagen.

```

4 usages  Alberto Brandon Velázquez
public class Error {
    3 usages
    private HttpStatus httpStatus;
    2 usages
    private String message;

    2 usages  Alberto Brandon Velázquez
    public Error(HttpStatus httpStatus, String message) {
        this.httpStatus = httpStatus;
        this.message = message;
    }

    2 usages  Alberto Brandon Velázquez
    public ResponseEntity GetResponseMessage() {
        Map<String, String> errorResponse = new HashMap<>();
        errorResponse.put("message", this.message);
        errorResponse.put("status", this.httpStatus.BAD_REQUEST.toString());
        ResponseEntity responseEntity = new ResponseEntity(errorResponse, this.httpStatus);
        return responseEntity;
    }
}

```

Imagen 14. Clase de error generada. Fuente: elaboración propia, app IntelliJ IDEA (2022)

- También se generaron clases para obtener todos los datos pasados por POST, en este caso solo se han usado para las consultas sobre genes y variantes.

```

package org.uichuimi.database.controllers.variants.utils;

import java.util.List;

2 usages  Alberto Brandon Velázquez
public class PostGenes {
    1 usage
    List<Long> ids;

    2 usages  Alberto Brandon Velázquez
    public List<Long> getIds() {
        return ids;
    }
}

```

Imagen 15. Clase para la obtención de datos por POST de genes. Fuente: elaboración propia, app IntelliJ IDEA (2022)

```

package org.uichuimi.database.controllers.variants.utils;

import java.util.List;

2 usages  Alberto Brandon Velázquez *
public class PostVariants {
    1 usage
    Long start;
    1 usage
    Long end;
    1 usage
    Double gmaf;
    1 usage
    Double polyphen;
    1 usage
    Double sift;
    1 usage
    List<Long> chromosomes;
    1 usage
    List<Long> genes;
    1 usage
    List<Long> effects;
    1 usage
    List<Long> impacts;
    1 usage
    List<Long> biotypes;
    1 usage
    List<GenotypeFilter> genotypeFilters;
    1 usage
    List<String> identifiers;

```

Imagen 16. Clase para la obtención de datos por POST de variantes. Fuente: elaboración propia, app IntelliJ IDEA (2022)

- Por otra parte, se generó una clase para almacenar todos los filtros de *genotypes* dado que de esta manera nos ahorra pasar una cantidad de parámetros al controlador.

```

package org.uichuimi.database.controllers.variants.utils;

import java.util.List;

7 usages  Alberto Brandon Velázquez
public class GenotypeFilter {
    1 usage
    List<Long> individual;
    1 usage
    List<Long> genotypeType;
    4 usages
    String selector;
    6 usages
    Integer number;

    3 usages  Alberto Brandon Velázquez
    public List<Long> getIndividual() { return individual; }

    2 usages  Alberto Brandon Velázquez
    public List<Long> getGenotypeType() { return genotypeType; }

    4 usages  Alberto Brandon Velázquez
    public String getSelector() {
        if (selector == null) {
            selector = "ALL";
        }
        if (number == null) {
            number = 1;
        }
        if (getIndividual().size() < number) {
            selector = "ALL";
        }
    }
}

```

Imagen 17. Clase de genotipos. Fuente: elaboración propia, app IntelliJ IDEA (2022)



Una vez familiarizados con el proyecto existente, se procedió entonces a realizar las optimizaciones de las funcionalidades ya existentes, como la creación de las nuevas funcionalidades pedidas por el cliente.

### 3.- Generar los modelos

Por cada una de las tablas que hay en la base de datos, se tuvo que generar una clase para poder generar el modelo de datos.

```
package org.uichuimi.database.models.variants;

import ...

12 usages  ▲ Pascual Lorente
@Entity(name = "populations")
@Entity(name = "population")
public class Population {

    6 usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;

    6 usages
    @Column(length = 8, unique = true, nullable = false)
    private String code;

    6 usages
    @Column(nullable = false)
    private String name;

    ▲ Pascual Lorente
    public Population() {
    }

    ▲ Pascual Lorente
    public Population(Long id, String code, String name) {
        this.id = id;
        this.code = code;
        this.name = name;
    }

    ▲ Pascual Lorente
    public Long getId() { return id; }

    ▲ Pascual Lorente
    public String getCode() { return code; }

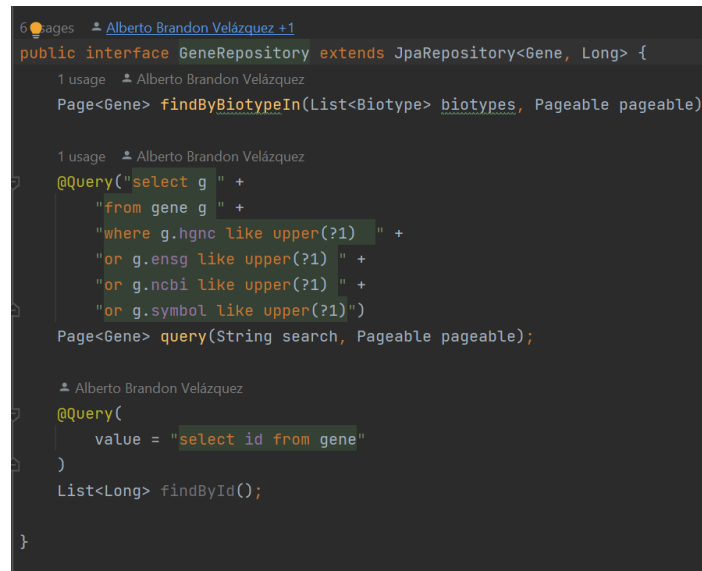
    ▲ Pascual Lorente
    public String getName() { return name; }

    ▲ Pascual Lorente
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Population that = (Population) o;
        return Objects.equals(id, that.id) && Objects.equals(code, that.code) && Objects.equals(name, that.name);
    }
}
```

Imagen 18. Modelo de genes. Fuente: elaboración propia, app IntelliJ IDEA (2022)

#### 4.- Generar los repositorios

Esta parte fue la más sencilla, dado que el ORM nos facilitaba el trabajo a la hora de realizar las consultas correspondientes, a continuación, veremos una imagen de ejemplo del repositorio de genes donde nosotros realizamos la sql que queremos.



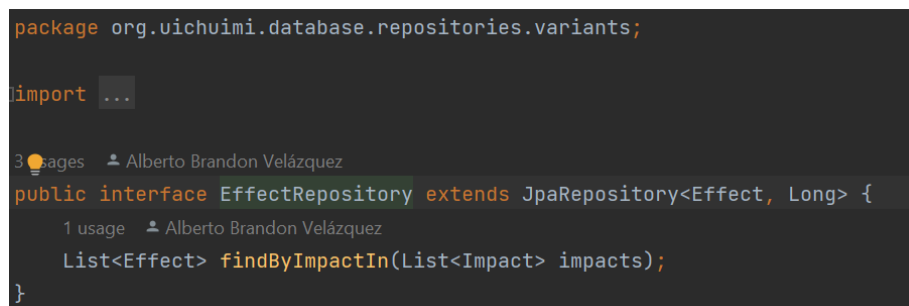
```
6 pages Alberto Brandon Velázquez +1
public interface GeneRepository extends JpaRepository<Gene, Long> {
    1 usage Alberto Brandon Velázquez
    Page<Gene> findByBiotypeIn(List<Biotype> biotypes, Pageable pageable);

    1 usage Alberto Brandon Velázquez
    @Query("select g " +
        "from gene g " +
        "where g.hgnc like upper(?1) " +
        "or g.ensg like upper(?1) " +
        "or g.ncbi like upper(?1) " +
        "or g.symbol like upper(?1)")
    Page<Gene> query(String search, Pageable pageable);

    Alberto Brandon Velázquez
    @Query(
        value = "select id from gene"
    )
    List<Long> findById();
}
```

Imagen 19. Repositorio de genes. Fuente: elaboración propia, app IntelliJ IDEA (2022)

A continuación, veremos un ejemplo de cómo lo realiza el ORM.



```
package org.vichuimi.database.repositories.variants;

import ...

3 pages Alberto Brandon Velázquez
public interface EffectRepository extends JpaRepository<Effect, Long> {
    1 usage Alberto Brandon Velázquez
    List<Effect> findByImpactIn(List<Impact> impacts);
}
```

Imagen 20. Repositorio de effects. Fuente: elaboración propia, app IntelliJ IDEA (2022)

Como podemos ver las automatizaciones que genera el ORM ayudan bastante a la hora de realizar consultas a la base de datos.

#### 5.- Generar los controladores

Para acabar esta fase se generaron los controladores, para las peticiones por GET o por POST de la siguiente manera.

```

/** Biotypes con identificador {id} */
Alberto Brandon Velázquez
@GetMapping("/{id}")
public ResponseEntity <Biotype> getBiotype(@PathVariable ("id") Long id){
    return ResponseEntity.ok(biotypeRepository.findById(id).get());
}

/** Genes con identificador {id} */
Alberto Brandon Velázquez
@PostMapping("/genes")
public ResponseEntity <List<Gene>> postGene(@RequestBody PostGenes body){
    if (body.getIds().size() > 200) {
        return new Error(HttpStatus.BAD_REQUEST, message: "El tamaño no debe superar los 200 elementos").getResponseMessage();
    }
    return ResponseEntity.ok(geneRepository.findAllById(body.getIds()));
}

```

Imagen 21. Controladores de genes Fuente: elaboración propia, app IntelliJ IDEA (2022)

## Segunda fase – Implementación del repositorio de variantes

Dado que el modelo de programación que permite mapear las estructuras de una base de datos relacional, sobre una estructura lógica de entidades con el objetivo de simplificar y acelerar el desarrollo de nuestra aplicación no estaba dando todo su fruto. Se decidió implementar el repositorio de variantes para que este hiciera lo que realmente queríamos. Por eso se desarrolló una clase cuyo método genera la consulta que realmente nos interesaba olvidándonos un poco del Mapeo Objeto-Relacional o como se conocen comúnmente, ORM (del inglés *Object Relational Mapping*).

Para esto se generó un método que se encarga de generar la query, haciendo dentro de ello los JOIN correspondientes a cada tabla según fuera necesario. Una vez se generaban los JOIN pasamos a generar el listado de predicados correspondientes para que buscará por los filtros pasados por el usuario en el POST.

Una vez realizado lo anterior, se extraen los datos correspondientes con los parámetros pasados por el usuario en forma de JSON. A continuación, se muestran unos trozos de código para que se vea como se ha implementado.

```

1 usage Alberto Brandon Velázquez
@Override
public Page<Long> findVariants(List<Long> genes, Long start, Long end, List<Long> chromosomes, List<Long> biotypes, Double gmaf, Double polyphen, Double sift, List<Long>

CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaQuery<Tuple> query = cb.createTupleQuery();
Root<Variant> variant = query.from(Variant.class);

//Generación de los JOIN más usados
Join<Variant, Consequence> consequences = null;
Join<Consequence, Transcript> transcripts = null;

//Cosas que van en el WHERE
List<Predicate> predicates = new ArrayList<>();
if (start != null) {
    predicates.add(cb.ge(variant.get("position"), start));
}
if (end != null) {
    predicates.add(cb.le(variant.get("position"), end));
}

```

Imagen 22. Implementación del repositorio de variantes. Fuente: elaboración propia, app IntelliJ IDEA (2022)

```
// Asigna el where de la query
query = query.where(predicates.toArray(new Predicate[0]));
// Prepara la query para contar el total de registros
CriteriaQuery<Tuple> countQuery = query.multiselect(cb.countDistinct(variant));
Tuple numregtuple = entityManager.createQuery(countQuery).getSingleResult();
Long numreg = (Long) numregtuple.get(0);
// Prepara query para extraer los datos
query = query.multiselect(variant.get("id"), variant.get("chromosome").get("sequence"), variant.get("position")).distinct(true);
query.orderBy(cb.asc(variant.get("chromosome").get("sequence")), cb.asc(variant.get("position")));
TypedQuery<Tuple> managerQuery = entityManager.createQuery(query);
List<Tuple> resultListTuples = managerQuery.setFirstResult(pageable.getPageNumber() * pageable.getPageSize()).setMaxResults(pageable.getPageSize()).getResultList();
// Convertir la lista de tuplas en una lista de variant.id
List<Long> resultList = new ArrayList<>();
resultListTuples.forEach(t -> resultList.add((Long) t.get(0)));
return new PageImpl<Long>(resultList, pageable, numreg);
```

Imagen 23. Implementación del repositorio de variantes. Fuente: elaboración propia, app IntelliJ IDEA (2022)

## 1.- Mejora de la paginación

Debido al gran tamaño de la base de datos, y con el fin de reducir el número de peticiones que se hacen al servidor de la RestAPI, las variantes se cargan en páginas del tamaño que el usuario pasa por parámetro (Sólo admite valores  $\leq 200$ ) o por defecto de 200 elementos desde el servidor.

## Tercera fase – Modificaciones para simplificar el JSON

Uno de los objetivos solicitados fue la reducción de elementos al realizar una consulta a la tabla de variantes, por eso se optó por realizar una proyección de la clase del modelo de datos para obtener los campos que eran de especial interés. Como podemos ver a continuación, con *@Projection* se realiza la proyección sobre la clase de variantes, donde en dicha clase hay que añadir los *@Value* para obtener el campo de interés con el método *getter* generado en cada modelo de datos.

```

package org.uichuimi.database.models.variants;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.data.rest.core.config.Projection;

import java.util.List;

6 usages  Alberto Brandon Velázquez
@Projection(types = Variant.class, name = "customVariant")
public interface VariantProjection {

    Alberto Brandon Velázquez
    @Value("#{target.id}")
    Long getId();

    Alberto Brandon Velázquez
    @Value("#{target.chromosome.id}")
    Long getChromosome();

    Alberto Brandon Velázquez
    Long getPosition();

    Alberto Brandon Velázquez
    String getReference();

    Alberto Brandon Velázquez
    String getAlternative();

    Alberto Brandon Velázquez
    String getIdentifier();

```

*Imagen 24. Proyección de la clase de variantes. Fuente: elaboración propia, app IntelliJ IDEA (2022)*

## Cuarta fase – Modificar el controlador de variantes

Para esta fase final, se decidió generar una clase que recogiera todos los parámetros que el usuario nos pasara por POST, quedando dicha clase de la siguiente manera.

```
public class PostVariants {  
  
    Long start;  
  
    Long end;  
  
    Double gmaf;  
  
    Double polyphen;  
  
    Double sift;  
  
    List<Long> chromosomes;  
  
    List<Long> genes;  
  
    List<Long> effects;  
  
    List<Long> impacts;  
  
    List<Long> biotypes;  
  
    List<GenotypeFilter> genotypeFilters;  
  
    List<String> identifiers;  
  
    Integer page;  
  
    Integer size;  
}
```

*Imagen 25. Clase para la recogida de los parámetros enviados por POST. Fuente: elaboración propia, app IntelliJ IDEA (2022)*

Ha esta clase solo se le generaron los *getters* dado que el usuario no iba a realizar ninguna modificación. Solamente se pusieron unos valores por defecto para el número de página que se quiere mostrar y el tamaño de elementos que quiere contener dicha página, dichos valores solo se modificarán cuando el usuario no los introduzca.

```

1 usage  Alberto Brandon Velázquez
public Long getStart() { return start; }

1 usage  Alberto Brandon Velázquez
public Long getEnd() { return end; }

1 usage  Alberto Brandon Velázquez
public Double getGmaf() { return gmaf; }

1 usage  Alberto Brandon Velázquez
public List<Long> getEffects() { return effects; }

1 usage  Alberto Brandon Velázquez
public List<Long> getImpacts() { return impacts; }

1 usage  Alberto Brandon Velázquez
public List<Long> getChromosomes() { return chromosomes; }

1 usage  Alberto Brandon Velázquez
public List<Long> getGenes() { return genes; }

1 usage  Alberto Brandon Velázquez
public List<Long> getBiotypes() { return biotypes; }

2 usages  Alberto Brandon Velázquez
public List<String> getIdentifiers() { return identifiers; }

1 usage  Alberto Brandon Velázquez
public List<GenotypeFilter> getGenotypeFilters() { return genotypeFilters; }

1 usage  Alberto Brandon Velázquez
public Double getPolyphen() { return polyphen; }

1 usage  Alberto Brandon Velázquez
public Double getSift() { return sift; }

2 usages  Alberto Brandon Velázquez
public Integer getPage() {
    if (page == null) {
        return 0;
    }
    return page;
}

3 usages  Alberto Brandon Velázquez
public Integer getSize() {
    if (size == null) {
        return 20;
    }
    return size;
}

```

Imagen 26. Getters de la clase que recoge los parámetros enviados por POST. Fuente: elaboración propia, app IntelliJ IDEA (2022)

Una vez generada la clase, se realizaron las modificaciones en el controlador, pasándole como parámetro un objeto de la clase anteriormente generada.

```

/**
 * Lista completa de variantes
 */
@Alberto Brandon Velázquez
@PostMapping(value = "/variants")
public ResponseEntity<Page<VariantProjection>> postVariant(
    @RequestBody PostVariants requestBody) {
    // Comprobar si el tamaño de página no sobre pasa el límite
    if (requestBody.getSize() > 200) {
        return new Error(HttpStatus.BAD_REQUEST, message: "El tamaño de página no debe superar los 200 elementos").getResponseMessage();
    }
    // Llamada al método independiente de Identifiers
    if (requestBody.getIdentifiers() != null) {
        return ResponseEntity.ok(variantRepository.findByIdIn(requestBody.getIdentifiers(), PageRequest.of(requestBody.getPage(), requestBody.getSize())));
    }
    Page<Long> ids = variantRepository.findVariants(
        requestBody.getGenes(),
        requestBody.getStart(),
        requestBody.getEnd(),
        requestBody.getChromosomes(),
        requestBody.getBiotypes(),
        requestBody.getGmaf(),
        requestBody.getPolyphen(),
        requestBody.getSift(),
        requestBody.getEffects(),
        requestBody.getImpacts(),
        requestBody.getGenotypeFilters(),
        PageRequest.of(requestBody.getPage(), requestBody.getSize()));
    Page<VariantProjection> page = new PageImpl<>(variantRepository.findByIdIn(ids.getContent(), ids.getPageable(), ids.getTotalElements()));
    return ResponseEntity.ok(page);
}

```

Imagen 27. Controlador de variantes. Fuente: elaboración propia, app IntelliJ IDEA (2022)

Dentro de esta fase nos encontramos con el problema de reducir las consultas, por eso se tuvo que desarrollar en los repositorios el siguiente método con *@EntityGraph*, en resumidas cuentas, lo que conseguimos con esto es cargar todo el grafo para hacer una única query y así evitar las queries de las relaciones asociadas.

```

1 usage @Alberto Brandon Velázquez
@EntityGraph(attributePaths = {"chromosome", "frequencies", "consequences", "genotypes", "consequences.transcript", "consequences.effect"})
List<VariantProjection> findByIdIn(List<Long> ids);

```

Imagen 28. Uso de *EntityGraph* en la proyección de variantes. Fuente: elaboración propia, app IntelliJ IDEA (2022)

## Cuarta fase – Testeo del proyecto



Imagen 29. Esquema de la realización de test del proyecto. Fuente: elaboración propia, app Excalidraw (2022)



Para realizar los test en el proyecto he decidido generar una colección de test en postman como veremos en la siguiente imagen.

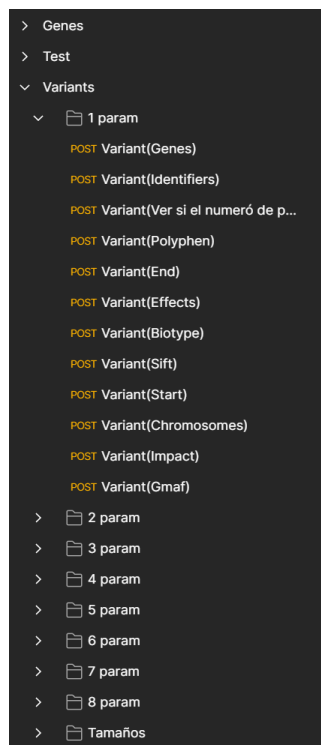


Imagen 30. Test del proyecto. Fuente: elaboración propia, app Postman.

Cada una de estas pruebas tiene un JSON que es enviado por GET o por POST como podemos ver en la siguiente imagen.

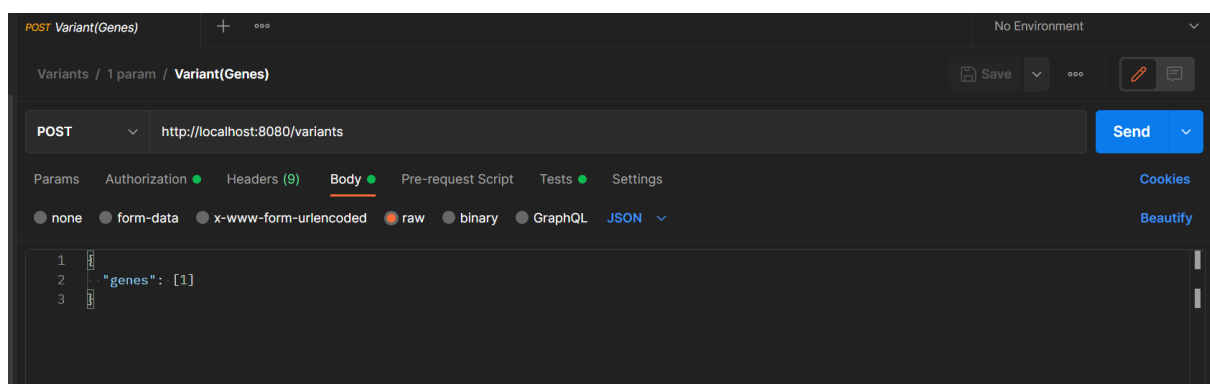


Imagen 31. Ejemplo de JSON. Fuente: elaboración propia, app Postman

Cada uno de estos ficheros contiene los siguientes test: Uno que mira si la respuesta del servidor es de tipo 200 (exitosa), otro que mira que la respuesta del servidor tarda menos de

700 ms, otro que comprueba si el número total de elementos es correcto y el último mide el número total de páginas de dicha consulta. A continuación, dejo la siguiente imagen de los test:

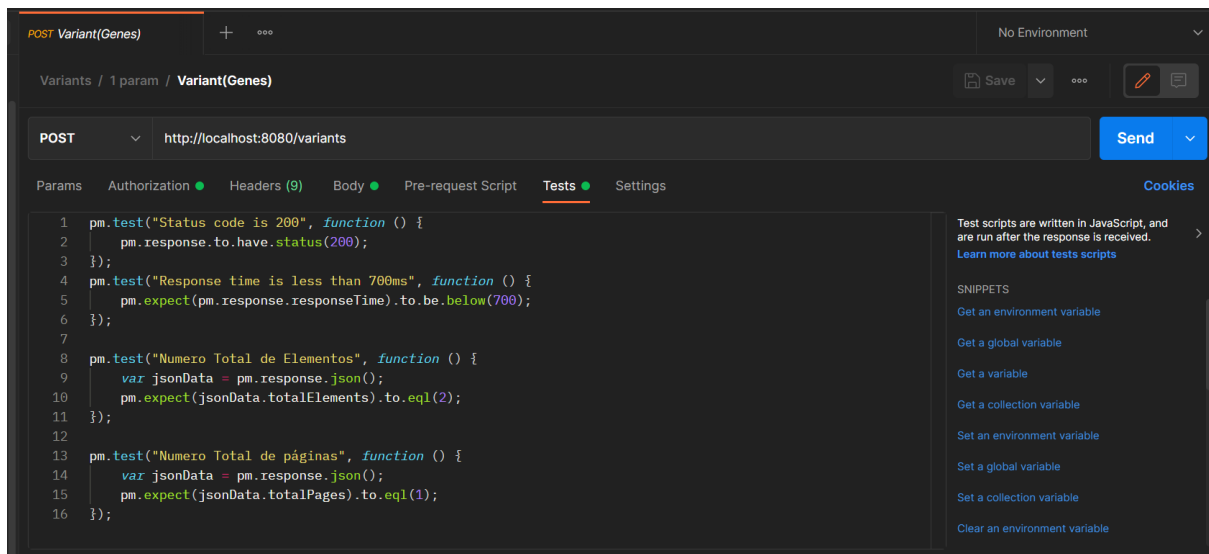


Imagen 32. Test que se realizan. Fuente: elaboración propia, app Postman

Finalmente, para la obtención de los tiempos ejecutamos la colección con 1000 iteraciones con un retardo de 0ms. Una vez finalizada la ejecución, Postman nos deja un JSON que contiene todos los tiempos de cada consulta. Gracias a esto se ha podido medir el rendimiento del aplicativo durante todo el desarrollo.

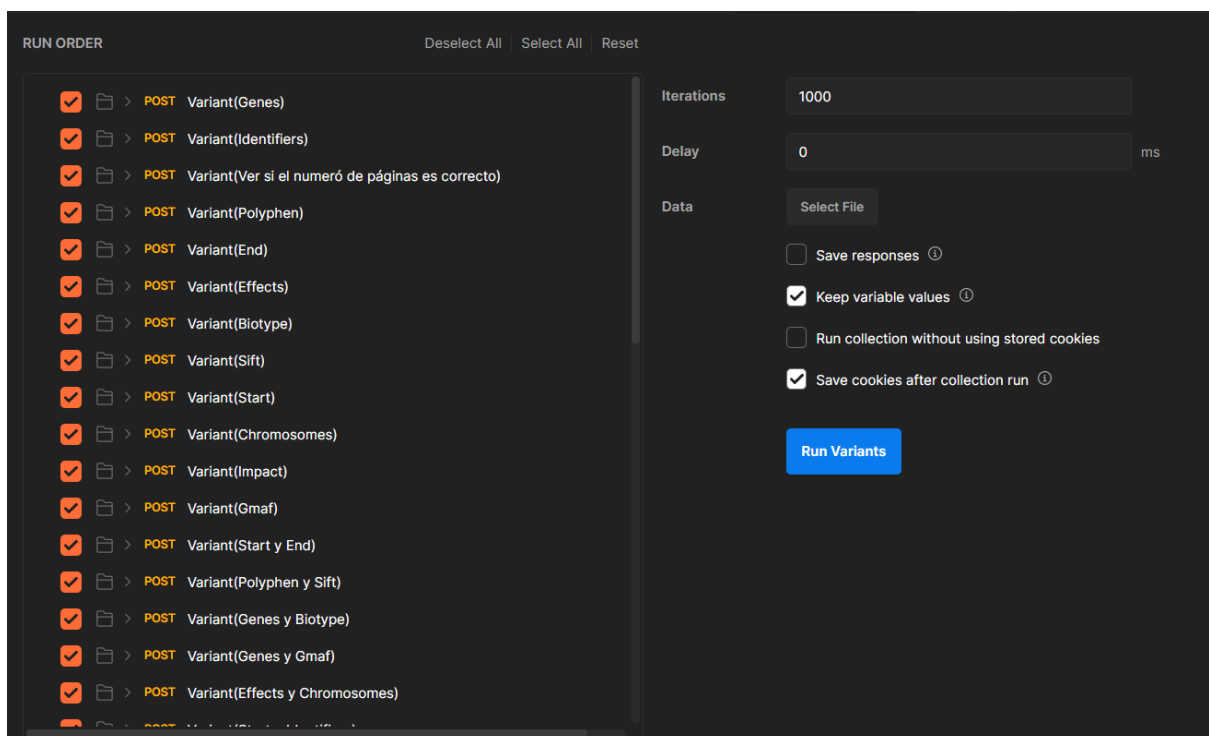


Imagen 33. Ejemplo de ejecución de test. Fuente: elaboración propia, app Postman

## 4.5 Tipos de resultados

Los resultados que se obtienen al realizar una petición GET o POST, pueden ser de tres tipos: un objeto, una lista o una página. A continuación, definiré cada uno de ellos con algún ejemplo de cómo se vería al realizar la consulta.

### 4.5.1 Objetos

Un objeto se corresponde con una fila o registro en una tabla de la base de datos, es decir una sola entidad. El resultado será un diccionario clave=valor. Por ejemplo, el objeto de tipo genes se compone de (Long id, String ensg, String hgnc, String ncbi, String symbol, String name, Biotype biotype, List<Transcript> transcripts) y tiene como métodos los *getters*.

```
public Gene(Long id, String ensg, String hgnc, String ncbi, String symbol, String name, Biotype biotype, List<Transcript> transcripts) {
    this.id = id;
    this.ensg = ensg;
    this.hgnc = hgnc;
    this.ncbi = ncbi;
    this.symbol = symbol;
    this.name = name;
    this.biotype = biotype;
    this.transcripts = transcripts;
}

public Long getId() { return id; }

public List<Transcript> getTranscripts() {return transcripts;}

public String getName() { return name; }

public String getHgnc() { return hgnc; }

public String getNcbi() { return ncbi; }

public String getEnsg() { return ensg; }

public String getSymbol() { return symbol; }

public Biotype getBiotype() { return biotype; }
```

Imagen 34. Modelo de datos de la tabla de genes. Fuente: elaboración propia, app IntelliJ IDEA (2022)

Cuando realicemos la consulta GET <http://localhost:8080/genes/1> nos dará como resultado el objeto de tipo <Gene> con id=1.

```
[
  {
    "ensg": "ENSG00000223972",
    "hgnc": "HGNC:37102",
    "ncbi": "100287102",
    "symbol": "DDX11L1",
```

```
"name": "DEAD/H-box helicase 11 like 1 (pseudogene)",  
}  
]
```

*Imagen 35. Consulta del GET. Fuente: elaboración propia, app IntelliJ IDEA (2022)*

## 4.5.2 Lista

Las listas se utilizan cuando el resultado puede contener un número indeterminado (entre 0 e infinito) de registros. El resultado es una lista de diccionarios. Por ejemplo, cuando pedimos el listado de genes (`List<genes>`) nos debe devolver un conjunto de objetos de tipo `genes`.

```
[  
  {  
    "ensg": "ENSG00000223972",  
    "hgnc": "HGNC:37102",  
    "ncbi": "100287102",  
    "symbol": "DDX11L1",  
    "name": "DEAD/H-box helicase 11 like 1 (pseudogene)",  
  },  
  {  
    "ensg": "ENSG00000223973",  
    "hgnc": "HGNC:37102",  
    "ncbi": "100287102",  
    "symbol": "DDX11L1",  
    "name": "DEAD/H-box helicase 11 like 1 (pseudogene)",  
  }  
]
```

*Imagen 36. Listado de genes. Fuente: elaboración propia, app IntelliJ IDEA (2022)*

## 4.5.3 Páginas

La paginación es un objeto (diccionario clave=valor) cuyos campos son (página, tamaño, offset, etc) y contenido, que es una lista de objetos. A continuación, se describen dichos campos, que a su vez son métodos que existen dentro del paginado.

```
"pageable": {  
  "sort": {  
    "empty": false,  
    "sorted": true,  
    "unsorted": false  
  },  
  "offset": 0,  
  "pageNumber": 0,  
  "pageSize": 5,  
}
```

```

    "unpaged": false,
    "paged": true
  },
  "last": false,
  "totalPages": 4217,
  "totalElements": 21085,
  "size": 5,
  "number": 0,
  "sort": {
    "empty": false,
    "sorted": true,
    "unsorted": false
  },
  "first": true,
  "numberOfElements": 5,
  "empty": false
}

```

*Imagen 37. Campos de paginación. Fuente: elaboración propia, app IntelliJ IDEA (2022)*

#### ❖ **Size (int NúmeroDeElementos)**

Este método se encarga de poner la cantidad de objetos por página cómo le pasamos en el parámetro. Una manera muy útil para cuando quieres mostrarle a los usuarios un muestreo de variantes o genes.

#### ❖ **Page (int NúmeroDePágina)**

Cuando realizamos la llamada a este método, es para poder obtener el número de la página que queramos, porque si lo mezclamos con `size(4)` por ejemplo obtendremos unas páginas con 4 elementos cada una y a lo mejor lo que buscamos o lo que queremos mostrar está en otra página.

#### ❖ **Sort (campo, asc o desc)**

La ordenación en la paginación se puede realizar de manera ascendente (asc) o descendente (desc) según el campo que le pasemos. paginación viene siendo una versión similar a la de las listas, pero dando esa mejoría visual a la persona que necesita leer una gran cantidad de datos.

## 4.6 Genes

Anteriormente la obtención de los genes se podía realizar de dos maneras distintas, una mediante la búsqueda completa (trayendo toda la información de cada gen) y otra por el identificador del gen.

Finalmente hemos optado por la opción más general, con un solo parámetro *search* para hacer una búsqueda de texto completa.

Por lo tanto, tenemos:

1. `/genes/{id}`, que devuelve un objeto de tipo Gen
2. `/genes&search={text}`, que devuelve una página de objetos de tipo Gen

La ruta `/genes/{id}` tiene un solo parámetro:

Parámetro	Tipo	Descripción
id	long	identificador del gen

Tabla 1. Parámetros que pasan a `genes/{id}`. Fuente: elaboración propia

La ruta `/genes&search={text}` tiene los siguientes parámetros:

Parámetro	Tipo	Descripción
search	string	texto para buscar, ignorando mayúsculas o minúsculas en los campos <i>symbol</i> , <i>ensg</i> , <i>hgnc</i> , <i>name</i> y <i>ncbi</i>
page	int	número de página (por defecto, 0)
size	int	tamaño de página (por defecto, 20)
sort	string	criterio de orden (por defecto, id)

Tabla 2. Parámetros de *genes* o que se pasan a *genes*. Fuente: elaboración propia

A continuación, veremos los campos del modelo de datos de genes, formado por: id, symbol, name, ensg, hgnc, ncbi y transcripts.

Campo	Tipo	Descripción
id	long	identificador interno de la base de datos
symbol	string	abreviatura común del nombre
name	string	nombre completo del gen
ensg	string	identificador en Ensembl
hgnc	string	identificador en HGCN
ncbi	string	identificador en NCBI
transcripts	[transcript]	lista de transcritos asociados al gen

Tabla 3. Campos de la clase de *genes*. Fuente: elaboración propia

Consulta: <http://localhost:8080/genes>

```
[
  {
    "id": 8244,
    "ensg": "ENSG00000070476",
    "hgnc": "HGNC:28160",
    "ncbi": "79364",
    "symbol": "ZXDC",
    "name": "ZXD family zinc finger C",
    "biotype": {
      "id": 4,
      "accession": "SO:0000010",
      "name": "protein_coding",
      "description": "A gene which, when transcribed, can be translated into a
protein."
    }
  },
  {
    "id": 40835,
    "ensg": "ENSG00000198455",
    "hgnc": "HGNC:13199",
    "ncbi": "158586",
    "symbol": "ZXDB",
    "name": "zinc finger X-linked duplicated B",
    "biotype": {
      "id": 4,
      "accession": "SO:0000010",
      "name": "protein_coding",
      "description": "A gene which, when transcribed, can be translated into a
protein."
    }
  }
]
```

*Cuadro 2. Resultados de la consulta de genes. Fuente: elaboración propia.*

**Consulta:** <http://localhost:8080/genes/1>

```
{
  "ensg": "ENSG00000223972",
  "hgnc": "HGNC:37102",
  "ncbi": "100287102",
  "symbol": "DDX11L1",
  "name": "DEAD/H-box helicase 11 like 1 (pseudogene)",
  "_links": {
    "self": {
      "href": "http://localhost:8080/genes/1"
    },
    "gene": {
      "href": "http://localhost:8080/genes/1"
    },
    "biotype": {
      "href": "http://localhost:8080/genes/1/biotype"
    }
  }
}
```

*Cuadro 3. Resultado de la consulta de genes/{id}. Fuente: elaboración propia*

## 4.7 BioTypes

No recibe ningún parámetro. Cuando se llame al método este dará un listado de biotypes con (id, accession), como en el siguiente ejemplo.

**Consulta:** <http://localhost:8080/biotypes>

```
[
  {
    "id": 21,
    "accession": "SO:0002103",
    "name": "TR_V_pseudogene",
    "description": "A pseudogenic variable region which closely resembles a known functional T receptor variable gene but in which the coding region has stop codons, frameshift mutations or a mutation that effects the initiation codon that rearranges at the DNA level and codes the variable region of an immunoglobulin chain."
  },
  {
    "id": 15,
    "accession": "SO:0002102",
    "name": "IG_V_pseudogene",
    "description": "A pseudogenic variable region which closely resembles a known functional immunoglobulin variable gene but in which the coding region has stop codons, frameshift mutations or a mutation that effects the initiation codon that rearranges at the DNA level and codes the variable region of an immunoglobulin chain."
  }
]
```

*Cuadro 4. Resultado de la consulta de Biotypes. Fuente: elaboración propia*

A continuación, veremos los campos del modelo de datos de *biotypes*, formado por: accession, name, description e id.

Campo	Tipo	Descripción
accession	string	Identificador SO:0000043
name	string	nombre corto (usado también en los VCF)
description	string	Descripción del biotipo
id	long	identificador en la base datos

*Tabla 4. Campos de la clase Biotypes. Fuente: elaboración propia*



## 4.8 Chromosomes

No recibe ningún parámetro. Cuando se llame al método este dará una lista de *chromosomes* con (*id*, *sequence*, *ncbi*, *genebank*, *refseq*, *ucsc*, *name*, *length*), como en el siguiente ejemplo.

**Ejemplo:** <http://localhost:8080/chromosomes>

```
{
  "id": 1,
  "ncbi": "1",
  "genebank": "CM000663.2",
  "refseq": "NC_000001.11",
  "ucsc": "chr1",
  "sequence": 1,
  "length": 248956422,
  "name": "chr1"
}, {
  "id": 2,
  "ncbi": "2",
  "genebank": "CM000664.2",
  "refseq": "NC_000002.12",
  "ucsc": "chr2",
  "sequence": 2,
  "length": 242193529,
  "name": "chr2"
}
```

Cuadro 5. Consulta de *chromosomes*. Fuente: elaboración propia

A continuación, veremos los campos del modelo de datos de *chromosomes*, formado por: *genebank*, *ncbi*, *refseq*, *sequence*, *ucsc*, *length* e *id*.

Campo	Tipo	Opciones	Descripción
genebank	String	CM000663.2, CM000664.2, ...	Identificador en genebank[1]
ncbi	String	1, 2, 3, ..., X, Y, MT	Identificador del NCBI[2]
refseq	String	NC_000001.11, NC_000002.12, NC_000003.12	Conjunto de secuencias completo [4]
sequence	int	orden natural	Código de referencia
ucsc	String	chr1, chr2, chr3, ..., chrX, chrY, chrM	Identificador de UCSC Genome Browser [3]
length	long	248956422	Tamaño de la cadena
id	long	1, 2, 3, 4, ..., 500	Identificador en la Base de Datos

Tabla 5. Campos de la clase *chromosomes*. Fuente: elaboración propia

## 4.9 Variants

Puede recibir parámetros (List<Long> genes\_ids, Long position\_start, Long position\_end, List<Long> chromosomes\_ids, List<Long> biotypes\_id, Double gmaf, Double polyphen, Double sift, List<Long> effects\_id, List<Long> impacts\_id, int size\_page). Retorna una lista de variantes paginada.

A continuación, veremos los campos del modelo de datos de *variants*, formado por: id, chromosoma\_id, position, reference, alternative e identifier.

Campo	Tipo	Opciones	Descripción
id	int	1, 2, 3, 4...	id de la variante
chromosome_id	int	1, 2, ..., 24, 25, ...	Id del cromosoma
position	int	1, 2, ..., 24212414, 24214378, 24306905, ...	Posición del cromosoma
reference	String	A, G, T, ...	Alelo de referencia
alternative	String	C, G, A, TATA, ...	Alelo mutante
identifier	String	rs10794659, rs672138, rs2272935, ...	identificador estándar

Tabla 6. Campos de la clase *Variants*. Fuente: elaboración propia

## 4.10 Populations

No recibe parámetros, devuelve una lista de *populations* (*id*, *code*, *name*).

**Consulta:** <http://localhost:8080/populations>

```
[
  {
    "id": 1,
    "code": "gca",
    "name": "Gran Canaria"
  },
  {
    "id": 2,
    "code": "all",
    "name": "Global"
  }
]
```

Cuadro 6. Consulta de *populations* (*id*, *code*, *name*). Fuente: elaboración propia

Consulta: <http://localhost:8080/populations/1>

```
{
  "id": 1,
  "code": "gca",
  "name": "Gran Canaria"
}
```

Cuadro 7. Consulta de *populations* (*id*, *code*, *name*). Fuente: elaboración propia

A continuación, veremos los campos del modelo de datos de *populations*, formado por: *code* y *name*.

Campo	Tipo	Opciones	Descripción
code	String	gca, all	Código de referencia
name	String	Global, Gran Canaria	Nombre

Tabla 7. Campos de la clase *chromosomes*. Fuente: elaboración propia

## 4.11 Individuals

No tiene parámetros, devuelve una lista de individuos (*id*, *code*).

Consulta: <http://localhost:8080/individuals>

```
[
  {
    "id": 9,
    "code": "CCAR_0283"
  },
  {
    "id": 19,
    "code": "CCAR_098"
  }
]
```

Cuadro 8. Consulta de *individuals*. Fuente: elaboración propia

A continuación, veremos los campos del modelo de datos de *individuals*, formado por: *code*.

Campo	Tipo	Opciones	Descripción
code	String	WDH21_002	Código de referencia

Tabla 8. Campos de la clase *individuals*. Fuente: elaboración propia

### 4.11.1 Impacts

No tiene parámetros, devuelve una lista de *impacts* (*id*, *name*, *sequence*).

Consulta: <http://localhost:8080/impacts>

```
[
  {
    "id": 1,
    "name": "MODIFIER",
    "sequence": 0
  },
  {
    "id": 2,
    "name": "LOW",
    "sequence": 1
  },
  {
    "id": 3,
    "name": "MODERATE",
    "sequence": 2
  },
  {
    "id": 4,
    "name": "HIGH",
    "sequence": 3
  }
]
```

Cuadro 9. Consulta de impact.. Fuente: elaboración propia

A continuación, veremos los campos del modelo de datos de *impacts*, formado por: name y sequence.

Campo	Tipo	Opciones	Descripción
name	String	MODIFIER, MODERATE	Nombre
sequence	int	0,1,2 ...	Código de referencia

Tabla 9. Campos de la clase impact. Fuente: elaboración propia

### 4.11.2 Effects

Recibe parámetros (*List<Impacts>*). Devuelve una lista de *effects*.

Consulta: <http://localhost:8080/effects>

```
[
  {
    "id": 6,
    "name": "A splice variant that changes the 2 base pair region at the 5' end of an intron.",
    "accession": "SO:0001575",
```

```

    "description": "splice_donor_variant",
    "impact": {
      "id": 4,
      "name": "HIGH",
      "sequence": 3
    }
  },
  {
    "id": 8,
    "name": "A sequence variant, that changes one or more bases, resulting in a
different amino acid sequence but where the length is preserved.",
    "accession": "SO:0001583",
    "description": "missense_variant",
    "impact": {
      "id": 3,
      "name": "MODERATE",
      "sequence": 2
    }
  }
]

```

Cuadro 10. Consulta de Effect. Fuente: elaboración propia

Consulta: <http://localhost:8080/effects/6>

```

{
  "id": 6,
  "name": "A splice variant that changes the 2 base pair region at the 5' end of
an intron.",
  "accession": "SO:0001575",
  "description": "splice_donor_variant",
  "impact": {
    "id": 4,
    "name": "HIGH",
    "sequence": 3
  }
}

```

Cuadro 11. Consulta de Effect. Fuente: elaboración propia

A continuación, veremos los campos del modelo de datos de *effects*, formado por: accession, description, name e impact.

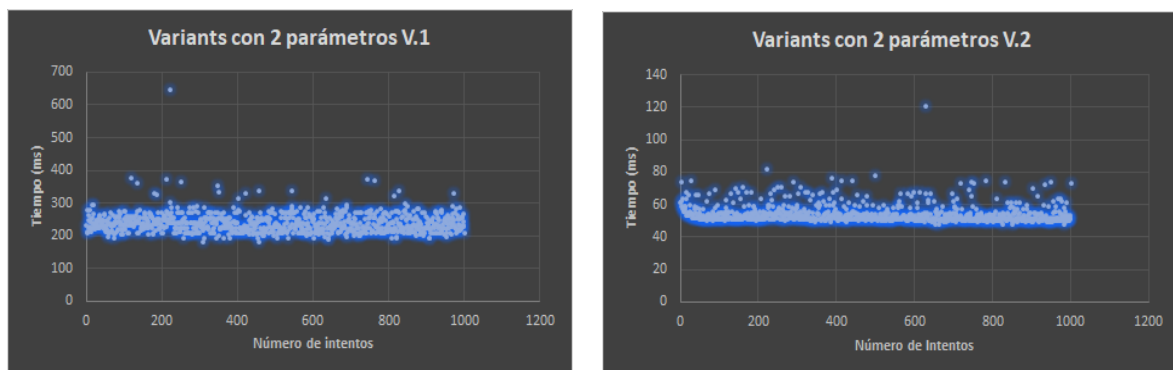
Campo	Tipo	Opciones	Descripción
accession	String	SO:0001575	Identificador SO: ...
description	String	splice_donor_variant	Descripción del effect
name	String	A splice variant that changes the 2 base pair region at the 5' end of an intron.	Nombre del effect
impact	impact		Impacto asociado al efecto

Tabla 10. Campos de la clase effect. Fuente: elaboración propia

# Capítulo 5

## Conclusión

Tras la puesta en marcha de todo el sistema de la aplicación web para variantes genéticas se ha podido observar una mejora sustancial en los tiempos de respuesta de la aplicación, dado que se han ido realizando pruebas de rendimiento desde el inicio hasta el final.



*Imagen 38. Comparación de tiempos de ejecución de variantes con 2 parámetros en el inicio del proyecto.  
Fuente: elaboración propia*

Se logró pasar de tiempos de espera de hasta 2-3 segundos para las peticiones más básicas de la aplicación, a respuestas de  $\pm 150$  milisegundos dependiendo del número de parámetros.

Adicionalmente, para algunas de las peticiones más complejas, como por ejemplo al aplicar el filtrado de variantes, así como los filtrados que ya existían en la aplicación, se logró que las peticiones más complejas tuvieran una media de tiempo de ejecución de  $\pm 350$  milisegundos, con ciertas variaciones o picos dependido de la combinación de filtros aplicados.

## 5.1 Trabajo futuro

Como trabajo para el futuro quedaría extender las opciones de devolución que se encuentran actualmente en la aplicación, ofreciendo más formatos en el cual descargar los datos de las variantes que se están observando en cada momento con sus filtros. Dado que actualmente solo se ofrece una versión JSON y podrían descargarse en formato de CSV o PDF.

Finalmente, se podría estudiar la posibilidad de cambiar el sistema que se usa actualmente para la base de datos en PostgreSQL a MySQL, para experimentar la existencia de posibles mejoras en el rendimiento de las peticiones a la BBDD y a la RestAPI gracias al uso del FULLTEXT que proporciona mysql, dado que FULLTEXT Index tiene mayores funcionalidades y mayor flexibilidad a la hora de buscar datos, teniendo disponibles funciones como contains, freetext, etc. También tiene un mejor rendimiento del que podrían obtener con búsquedas del tipo like '%xxx%'.

Este proyecto implicó un duro trabajo, dado que no sólo se trataba de la evolución y comprensión de código ajeno, sino también la combinación y sincronización de varias tecnologías desconocidas y la necesidad de trabajar dentro de un mundo del cuál no tenía mucho conocimiento como es el de la genética, variantes, etc.

Esta experiencia me sirve de preparación para el futuro, al ampliar conocimientos sobre las tecnologías usadas, pero también a nivel personal por la coordinación que he tenido que llevar con mis tutores durante este largo recorrido.

# Bibliografía

- Dbeaver Community (2022). Free Universal Database Tool. <https://dbeaver.io/>
- DeGregorio, A. (24 de diciembre de 2021). Like Queries in Spring JPA Repositories. *Baeldung*. <https://www.baeldung.com/spring-jpa-like-queries>
- Celniker, S.E., Wheeler, D.A., Kronmiller, B. et al. (23 de diciembre de 2002). Finishing a whole-genome shotgun: Release 3 of the *Drosophila melanogaster* euchromatic genome sequence. *BMC: Part of Springer Nature. Genome Biology*. <https://doi.org/10.1186/gb-2002-3-12-research0079>
- Centro Nacional de análisis genómico (2016). *Centre for Genomic Regulation (CRG). Centre nacional d'anàlisi genòmica (cnag)*. <https://www.cnag.crg.eu/>.
- Dbeaver (2022). *Página web DBeaver Community*. [Fotografía]. <https://dbeaver.io/>
- Escuela de Ingeniería Informática (2013). Objetivos y Competencias del GII. *Escuela de Ingeniería Informática (EII). Universidad de Las Palmas de Gran Canaria (ULPGC)*. [https://www.eii.ulpgc.es/tb\\_university\\_ex/?q=objetivos-y-competencias-del-gii](https://www.eii.ulpgc.es/tb_university_ex/?q=objetivos-y-competencias-del-gii)
- ESPAÑA CIO (2022). [Fotografía]. <https://www.ciospain.es/gobierno-ti/java-8-sigue-dominando-pero-se-acerca-la-ola-de-java-17>
- Excalidraw (2022). *Aplicación web para la realización de esquemas*. <https://excalidraw.com/>
- Spring (2022). *Página web spring home (Java)*. [Fotografía]. <https://spring.io/>
- Genome Browser (2022). Our tools. *University of California Santa Cruz (UCSC). Genomics Institute*. <https://genome.ucsc.edu/>
- Gierke, O., Darimont, T., Strobl, C. et al. (19 de marzo de 2022). Spring Data JPA-Reference documentation. *Spring*. <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#referencespring>
- gnomAD browser (2020). Genome Aggregation Database. <https://gnomad.broadinstitute.org/>
- Git (2022). *Página web git*. [Fotografía]. <https://git-scm.com/>
- GRC (2019). *The Genome Reference Consortium*. Genome Reference Consortium [GRC]. <https://www.ncbi.nlm.nih.gov/grc>
- Henríquez-González, J. (2017). Combinación y anotación de variantes genéticas. Universidad de Las Palmas de Gran Canaria (ULPGC). <https://accedacris.ulpgc.es/handle/10553/22629>



IntelliJ IDEA (2022). *Página web IntelliJ IDEA – IDE para JVM eficaz y ergonómico. JET BRAINS. [Fotografía]*. <https://www.jetbrains.com/es-es/idea/>

Krill, P. (2022, marzo). Java 8 sigue dominando, pero se acerca la ola de Java 17 [Fotografía].

Ley Orgánica 3/2018, de 5 de diciembre, de protección de Datos Personales y garantía de los derechos digitales. *Boletín Oficial del Estado de fecha de 6 de diciembre de 2018, n° 294, 119788-119857*. <https://www.boe.es/boe/dias/2018/12/06/pdfs/BOE-A-2018-16673.pdf>

Lorente-Arencibia P, et al. (2016). Evaluating the genetic diagnostic power of exome sequencing: Identifying missing data. *Cold Spring Harbor Laboratory (CSH)*. <https://doi.org/10.1101/068825>

McLaren, W., Gil, L., Hunt, S.E. et al. (2016). The ensemble Variant Effect Predictor. *Genome Biology* 17, 122. Recuperado de: <https://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-0974-4>

Müller-Wille, S. y Hall, H. (2016). Legumes and Linguistics: Translating Mendel For the Twenty-First Century. *The British Society for the History of Science (BSHS)*. <http://www.bshs.org.uk/bshs-translations/mendel>

National Library of Medicine (2021). GenBank Overview. *National Center for Biotechnology Information*. <https://www.ncbi.nlm.nih.gov/genbank/>

National Library of Medicine (2022). Welcome to NCBI. *National Center for Biotechnology Information*. <https://www.ncbi.nlm.nih.gov/>

National Library of Medicine (2022). RefSeq: NCBI Reference Sequence Database. A comprehensive, integrated, non-redundant, well-annotated set of reference sequences including genomic, transcript, and protein. *National Center for Biotechnology Information*. <https://www.ncbi.nlm.nih.gov/refseq/>

NIH (2019). Ácido desoxirribonucleico [ADN]. *National Human Genome Research Institute*. [www.genome.gov/es/about-genomics/fact-sheets/acido-desoxirribonucleico](http://www.genome.gov/es/about-genomics/fact-sheets/acido-desoxirribonucleico)

NIH (2022). ClinVar aggregates information about genomic variation and its relationship to human health. *National Library of Medicine. National Center for Biotechnology Information*. <https://www.ncbi.nlm.nih.gov/clinvar/>

OMIM (2022). Online Mendelian Inheritance in Man. An Online Catalog of Human Genes and Genetic Disorders. *Human Genetics Knowledge for the World*. <https://www.omim.org/>

PostgreSQL (2022). *Página web PostgreSQL: The world's most advanced open source relational database. [Fotografía]*. <https://www.postgresql.org/>

SNGULAR (2022). Postman I: Explorando la herramienta [Fotografía].  
<https://www.sngular.com/es/postman-i-comenzando-a-explorarlo/>

The Variant Call Format [VCF] (27 de julio de 2022) <https://samtools.github.io/hts-specs/VCFv4.2.pdf>

Wikipedia (20 de febrero de 2022). *Vacuna del ARN* [Fotografía].  
[https://es.wikipedia.org/wiki/Vacuna\\_de\\_ARN](https://es.wikipedia.org/wiki/Vacuna_de_ARN)

Wikipedia (19 de abril de 2022). *Ácido desoxirribonucleico* [Fotografía].  
[https://es.wikipedia.org/wiki/%C3%81cido\\_desoxirribonucleico](https://es.wikipedia.org/wiki/%C3%81cido_desoxirribonucleico)