UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño e implementación de una arquitectura de microservicios basada en modelos conceptuales para automatizar la clasificación de variaciones genéticas

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Conesa Sánchez, Lucas

Tutor/a: García Simón, Alberto

Cotutor/a: Pastor López, Oscar

Director/a Experimental: Costa Sánchez, Mireia

CURSO ACADÉMICO: 2023/2024

Resum

L'estudi del nostre ADN permet previndre, diagnosticar i tractar un gran ventall de malalties, contribuint a millorar la qualitat de vida de la societat. En aquest context, una de les àrees que més atenció ha rebut és l'estudi de com les variacions genètiques es relacionen amb diferents malalties, procés conegut com a classificació de variacions. Històricament, la classificació de variacions ha sigut un procés complex, subjectiu i ha requerit d'una gran tasca manual sense un suport tecnològic robust. Per tal de facilitar la tasca als experts del domini, aquest treball ha col·laborat en el disseny d'una arquitectura basada en microserveis que automatitze algunes de les tasques que es duen a terme durant el procés de classificació de variacions, ocultant la complexitat subjacent d'aquestes al usuari. En concret, s'han definit i implementat tres dels mòduls que componen aquesta arquitectura. En primer lloc, un microservei que permeta pujar, preparar i encriptar fitxers amb la informació genètica dels pacients per a que siguen analitzats. En segon lloc, un orquestrador que s'encarrega de que els diferents mòduls es coordinen i cooperen per analitzar les dades d'un pacient i generar un informe amb informació rellevant a nivell clínic. En tercer lloc, un microservei que permeta als usuaris de la plataforma descarregar de manera segura els informes clínics de cada pacient. Tant els tres mòduls desenvolupats en aquest treball com la resta de l'arquitectura han sigut validats en un entorn real amb oncòlegs que han proporcionat informació de pacients reals.

Paraules clau: Microserveis, Models Conceptuals, Genètica, JavaScript, Cloud

Resumen

El estudio de nuestro ADN permite prevenir, diagnosticar y tratar un gran abanico de enfermedades, contribuyendo a mejorar la calidad de vida de la sociedad. En este contexto, una de las áreas que más atención ha recibido es el estudio de cómo las variaciones genéticas se relacionan con distintas enfermedades, proceso conocido como clasificación de variaciones. Históricamente, la clasificación de variaciones ha sido un proceso complejo, subjetivo y ha requerido de una gran labor manual sin un soporte tecnológico robusto. Para intentar facilitar la labor a los expertos del dominio, este trabajo ha colaborado en el diseño de una arquitectura basada en microservicios que automatice algunas de las tareas que se dan durante el proceso de clasificación de variación, ocultando la complejidad subyacente de las mismas al usuario. En concreto, se han definido e implementado tres de los módulos que componen esta arquitectura. En primer lugar, un microservicio que permita subir, preparar y encriptar ficheros con la información genética de los pacientes para que sean analizados. En segundo lugar, un orquestador que se encarga de que los diferentes módulos se coordinen y cooperen para analizar los datos de un paciente y generar un reporte con información relevante a nivel clínico. En tercer lugar, un microservicio que permita a los usuarios de la plataforma descargar de una manera segura los reportes clínicos de cada paciente. Tanto los tres módulos desarrollados en este trabajo como el resto de la arquitectura han sido validados en un entorno real con oncólogos que han proporcionado información de pacientes reales.

Palabras clave: Microservicios, Modelos Conceptuales, Genética, JavaScript, Cloud

Abstract

Studying our DNA allows us to prevent, diagnose, and treat a wide range of diseases, contributing to improving the quality of life in society. In this context, one of the

areas that has received the most attention is the study of how genetic variations are related to different diseases, a process known as variant classification. Historically, variant classification has been a complex and subjective process requiring significant manual effort without robust technological support. To facilitate the work for domain experts, this project has contributed to designing a microservices-based architecture to automate some of the tasks involved in the variant classification process, thereby hiding the underlying complexity from the user. Specifically, three modules that comprise this architecture have been defined and implemented. First, a microservice that allows for the upload, preparation, and encryption of files containing patients' genetic information for analysis. Second, an orchestrator responsible for ensuring that the different modules coordinate and cooperate to analyze a patient's data and generate a report with clinically relevant information. Third, a microservice that enables users of the platform to securely download clinical reports for each patient. Both the three modules developed in this work and the rest of the architecture have been validated in a real-world environment with oncologists who provided information from real patients.

Key words: Microservices, Conceptual Models, Genetics, JavaScript, Cloud

Índice general

Apéndice

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Motivación

La medicina de precisión es uno de los enfoques existentes en la atención médica que se basa en la comprensión detallada de las características individuales de cada paciente, incluyendo su genética, ambiente y estilo de vida. La medicina de precisión busca personalizar el tratamiento y la prevención de enfermedades para cada persona, en lugar de aplicar enfoques generales que se ha comprobado que no son efectivos para todos. Por ejemplo, en un estudio publicado en la revista Nature Medicine en 2016, investigadores encontraron que el uso de terapias dirigidas por biomarcadores en pacientes con cáncer de pulmón avanzado mejoró significativamente la supervivencia en comparación con la quimioterapia estándar [1].

En el centro de la medicina de precisión yace el ácido desoxirribonucleico (ADN), una molécula que contiene la información genética esencial para el desarrollo, funcionamiento y reproducción de todos los organismos vivos. Está compuesto por dos cadenas entrelazadas que forman una estructura de doble hélice. Cada cadena está formada por una secuencia de nucleótidos, que contienen una base nitrogenada (adenina, timina, citosina o guanina), un azúcar (desoxirribosa) y un grupo fosfato. La secuencia de estas bases nitrogenadas, conocida como genoma, determina las instrucciones genéticas que guían la producción de proteínas, las cuales son fundamentales para las funciones biológicas en el organismo. El ADN se encuentra principalmente en el núcleo de las células y es transmitido de generación en generación, asegurando la continuidad de las características hereditarias. [2] [3] .

Dentro del genoma humano, se encuentra el gen, una región del ADN que contiene las instrucciones necesarias para la síntesis de una proteína particular, la regulación de la actividad de otras moléculas en el organismo, y muchos otros procesos. Los genes actúan como unidades funcionales fundamentales que determinan una variedad de características biológicas, incluyendo la predisposición a enfermedades y rasgos fenotípicos observables [4] .

La variabilidad genética es una característica destacada del genoma humano. Las diferencias individuales se manifiestan principalmente en diversas regiones del ADN, incluyendo los genes, presentándose como mutaciones, polimorfismos u otras variaciones en la secuencia de ADN. La variabilidad genética se refleja directamente en el proteoma, mientras que también influye indirectamente en el microbioma y el metaboloma. Estas variaciones en conjunto contribuyen a las diferencias individuales observadas. Esta diversidad genética es esencial para entender las diferencias en la susceptibilidad a enfermedades, las respuestas a tratamientos y las interacciones con el entorno [5].

En relación a esto, el fenotipo refleja la expresión observable de la información genética en respuesta a factores ambientales. Es el conjunto de características morfológicas, fisiológicas y conductuales de un organismo, que abarca desde características básicas como el color de los ojos hasta predisposiciones a enfermedades complejas. Así, el fenotipo resulta de la interacción dinámica entre el genotipo y el ambiente [6] [7].

Ahora que ya se conocen algunos términos clave sobre la medicina de precisión, se procederá a describir la importancia de esta y algunas de sus principales ventajas. La medicina de precisión representa un avance fundamental en el campo de la atención médica, ofreciendo un enfoque altamente personalizado que tiene el potencial de revolucionar la forma en que se diagnostican, previenen y tratan las enfermedades. Su importancia radica en varias áreas clave, cada una de las cuales contribuye significativamente a mejorar la salud y el bienestar de las personas.

En primer lugar, la capacidad de predecir la susceptibilidad de un individuo a ciertas enfermedades mediante el análisis de su perfil genético es una herramienta poderosa para la prevención y el manejo temprano de enfermedades. A través de técnicas avanzadas de secuenciación genética y análisis bioinformático, los médicos pueden identificar variantes genéticas asociadas con un mayor riesgo de desarrollar condiciones específicas, como enfermedades cardíacas, cáncer o trastornos metabólicos. Por ejemplo, estudios han demostrado que ciertas variantes genéticas están asociadas con un mayor riesgo de enfermedad cardíaca coronaria, lo que permite a los médicos tomar medidas preventivas, como cambios en el estilo de vida o el monitoreo regular, para reducir este riesgo [8].

En segundo lugar, la medicina de precisión permite la implementación de estrategias personalizadas de prevención que se adaptan específicamente al perfil genético y al estilo de vida de cada individuo. Esto puede incluir recomendaciones dietéticas, pautas de ejercicio y programas de detección temprana adaptados a las necesidades y riesgos únicos de cada paciente. Por ejemplo, en el caso de la diabetes tipo 2, los médicos pueden utilizar la información genética de un paciente para personalizar su plan de manejo, incluida la selección de medicamentos específicos y la modificación de la dieta y el ejercicio para mejorar el control glucémico y reducir el riesgo de complicaciones [9].

Finalmente, la medicina de precisión también facilita la selección de tratamientos más efectivos y personalizados para cada paciente, optimizando así los resultados terapéuticos y minimizando los efectos secundarios. Al comprender mejor la relación entre la genética de un individuo y su respuesta a ciertos medicamentos, los médicos pueden elegir terapias específicas que sean más susceptibles de ser eficaces y bien toleradas por el paciente. Por ejemplo, en el campo de la oncología, los análisis genéticos pueden ayudar a identificar mutaciones específicas en tumores que indican una mayor probabilidad de respuesta a ciertos tratamientos dirigidos, como la terapia dirigida a BRAF en pacientes con melanoma metastásico [10].

Dentro de la medicina de precisión, es importante considerar el concepto de significancia clínica, que se refiere al impacto médico de variaciones genéticas específicas en relación con la salud y el desarrollo de enfermedades. Algunas variaciones pueden ser benignas, mientras que otras pueden tener consecuencias significativas en el riesgo de enfermedad, la respuesta a tratamientos y el pronóstico. Los genetistas y otros expertos investigan estas relaciones utilizando una variedad de enfoques y técnicas para comprender cómo las variaciones genéticas afectan la salud humana [11] [12].

Una de las técnicas más destacadas a la hora de evaluar la significancia clínica, es el proceso de clasificación de variantes, el cual se encarga de categorizar las distintas variaciones genéticas en función de su potencial impacto en la salud humana. Este proceso suele seguir un flujo de trabajo estándar o 'pipeline'. Este proceso se compone de los siguientes pasos:

Anotación de variantes: Se identifican y etiquetan las variantes genéticas en el genoma de un individuo utilizando herramientas bioinformáticas especializadas. Esta etapa implica asignar funciones biológicas conocidas a las variantes identificadas [13].

Filtrado de variantes: Se eliminan las variantes que no tienen relevancia clínica o que no están asociadas con la enfermedad en cuestión. Esto se logra mediante la aplicación de criterios específicos, como la frecuencia en la población, la predicción de impacto funcional y la evidencia de asociación con la enfermedad [14].

Clasificación de variantes: Las variantes restantes se clasifican según su probabilidad de contribuir a la enfermedad. Este proceso implica evaluar la evidencia científica disponible, como estudios clínicos y bases de datos genómicas, para determinar la significancia clínica de cada variante. Este proceso se suele llevar a cabo mediante guías clínicas como la ACMG/AMP [15].

Generación de resultados: Finalmente, se generan informes que resumen las variantes identificadas y su relevancia clínica para el paciente. Estos informes proporcionan recomendaciones específicas para el manejo clínico, como opciones de tratamiento o medidas preventivas [16].

La clasificación de variantes es un componente esencial en el proceso de medicina de precisión, ya que permite discernir entre las numerosas variantes genéticas presentes en el genoma de un individuo y determinar cuáles tienen importancia clínica. Este proceso es crucial porque facilita la identificación de biomarcadores genéticos asociados con el riesgo de desarrollar enfermedades específicas, lo que posibilita la detección temprana y la aplicación de medidas preventivas. En resumen, la clasificación de variantes es fundamental en la medicina de precisión para traducir la información genética en conocimientos clínicos accionables, mejorando así la atención médica individualizada y la gestión de enfermedades.

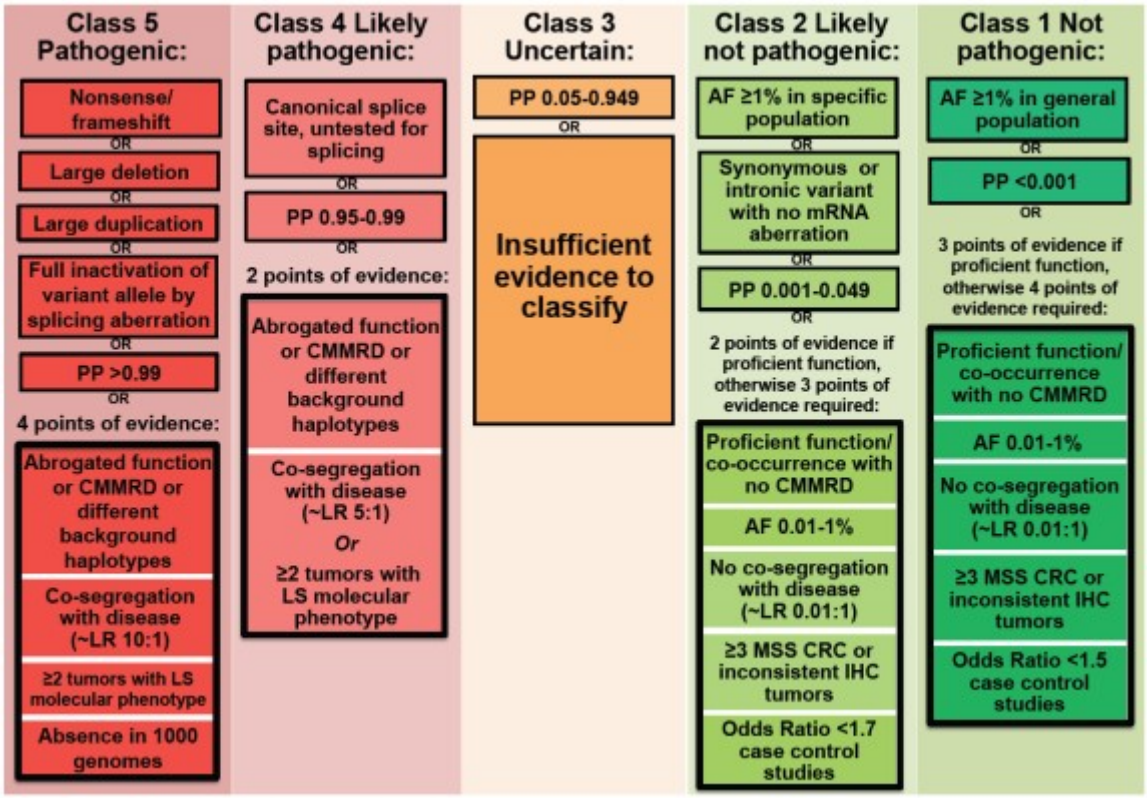

Figura 1.1: Ejemplo clasificación de variantes.

El presente Trabajo de Fin de Grado se ha llevado a cabo dentro del marco de un grupo de investigación especializado en genómica y medicina de precisión, donde el autor de este proyecto inició su participación mediante prácticas respaldadas por una beca de colaboración. En la actualidad, el autor continúa involucrado en este proyecto dentro del grupo de investigación. Este grupo se encuentra inmerso en el desarrollo de una *spin-off* dedicada a proporcionar un servicio integral de generación de informes clínicos a profesionales médicos y genetistas.

Este servicio requiere la creación de un punto de entrada específico, diseñado para la carga y procesamiento de archivos relevantes, con el propósito de simplificar y hacer transparente al usuario la complejidad inherente en la gestión de información de los pacientes. En este contexto, el objetivo central de este Trabajo de Fin de Grado radica en **el desarrollo y la implementación de dicho punto de entrada**.

La importancia de este proyecto reside en su capacidad para optimizar el flujo de trabajo dentro del ámbito médico-genético, al proporcionar una interfaz intuitiva y eficiente para la gestión de datos clínicos. Esto no solo facilitará la labor de los profesionales de la salud y genetistas, sino que también promoverá una atención médica más personalizada y precisa para los pacientes.

1.2 Objetivos

En este apartado se definirán los objetivos de este Trabajo Fin de Grado, el cual comienza por el objetivo principal.

Objetivo principal: Diseñar y desarrollar una plataforma que permita a médicos y genetistas realizar sus procesos de análisis de variaciones.

Cabe recalcar que se desarrollará una serie de microservicios los cuales se integrarán dentro de una arquitectura existente. Dicha arquitectura se mostrará más adelante inlcuyendo una explicación de cómo era antes y después de integrar los distintos microservicios.

Este objetivo principal ha sido dividido en tres objetivos específicos para facilitar su consecución.

Objetivo Específico 1: Entender el dominio de los usuarios. Entender qué tareas realizan los usuarios, cómo las realizan, qué problemas tienen y cómo les podemos ayudar a solucionarlos.

Objetivo Específico 2: Desarrollar la herramienta. Ampliar la plataforma ya existente, desarrollando e integrando nuevos módulos, componentes y funcionalidades.

Objetivo Específico 3: Probar y validar la herramienta. Reuniones con expertos y potenciales usuarios que probarán la herramienta y nos proporcionarán. *feedback*.

1.3 Metodología

Este Trabajo Fin de Grado sigue la metodología *Design Science* según Roel Wieringa [17]. Esta metodología se utiliza en diversas disciplinas para abordar problemas complejos y desarrollar soluciones innovadoras. Se fundamenta en la premisa de que el diseño es una actividad fundamental en la creación de artefactos que resuelven problemas prácticos en un contexto determinado. En resumen, Design Science busca estudiar cómo un

artefacto interactúa con un entorno determinado para evaluar si soluciona una problemática específica.

Primero, se definen las preguntas de investigación a partir de los objetivos específicos. Estas preguntas de investigación guían todo el proceso y permiten seleccionar la estrategia más adecuada. Estas preguntas están diseñadas para comprender el problema, identificar las necesidades del usuario y establecer una base sólida para la generación de soluciones. Las preguntas de ivnestigación que han guado la aplicación de *Design Science* son las siguientes:

Objetivo Específico 1: Entender el dominio de los usuarios.

RQ1 : ¿Cuál es el formato de los datos genómicos típicos?

RQ2 : ¿Cuáles son las características más relevantes a tener en cuenta al trabajar con datos genómicos?

RQ3 : ¿Cuál es la arquitectura más adecuada?

RQ4 : ¿Cuál es el estado actual de la plataforma a mejorar?

Objetivo Específico 2: Desarrollar la herramienta.

RQ5 : ¿Qué componentes principales debe tener la arquitectura?

RQ6 : ¿Qué stack tecnológico es el más adecuado?

RQ7 : ¿Cómo realizar el despliegue de la arquitectura?

Objetivo Específico 3: Probar y validar la herramienta.

RQ8: ¿En qué medida la herramienta resulta útil para los usuarios finales?

Primero, se deben identificar del artefacto a desarrollar y el contexto en el cual será creado En este caso:

ARTEFACTO Una plataforma web que se integrará en la arquitectura de microservicios existente.

CONTEXTO El proceso de creación de una *spin-off* dedicada a la medicina de precisión.

La identificación precisa del artefacto y su contexto resulta primordial, ya que define la estructura y el alcance del trabajo. Una vez identificados artefcacto y contexto, se lleva a cabo la identificación de los stakeholders, es decir, aquellos individuos, grupos o entidades con algún interés o afectados por los resultados, el proceso o las implicaciones del estudio. En este caso:

STAKEHOLDERS Los usuarios de esta plataforma web: médicos y genetistas. No obstante, de forma indirecta también incluye a los pacientes, quienes se beneficiarán del uso de la herramienta. También se identifican como stakeholders a los tutores de este trabajo, los miembros del grupo de investigación, y personas responsables de la validación de la herramienta.

En *Design Science*, se distinguen dos tipos principales de ciclos: el ciclo de diseño y el ciclo de conocimiento. Estos ciclos se complementan mutuamente y juntos contribuyen a la creación y validación de conocimientos y artefactos útiles. El ciclo de diseño se centra en la creación y mejora de artefactos, mientras que el ciclo de conocimiento se enfoca en la generación y validación de conocimientos científicos a través del estudio y análisis de

los artefactos diseñados. Ambos ciclos son iterativos y constan de diversas fases. Además están estrechamente interrelacionados y se retroalimentan mutuamente. El ciclo de diseño proporciona los artefactos necesarios para ser evaluados en el ciclo de conocimiento, mientras que el ciclo de conocimiento genera nuevos conocimientos y teorías que pueden influir y mejorar los futuros diseños. Esta interacción continua asegura que tanto los artefactos como los conocimientos científicos evolucionen y mejoren con el tiempo, contribuyendo al avance tanto de la práctica profesional como de la investigación científica.

Este Trabajo Fin de Grado sigue un ciclo de diseño cuyas fases se explicarán más adelante. Una vez identificados el artefacto, el contexto y los stakeholders, el siguiente paso consiste en definir el ciclo de diseño y las preguntas de investigación. Un ciclo de diseño comprende una serie de fases organizadas secuencialmente para concebir, desarrollar y finalizar un producto, servicio o solución de diseño. En este Trabajo Fin de Grado, se seguirán las siguientes fases:

Investigación del problema: Esta fase implica comprender tanto el contextocientífico asociado al problema como las necesidades y desafíos del usuario. Se lleva a cabo a través de la investigación del contexto, el análisis de datos y la definición clara del problema a resolver.

Diseño de la solución: Aquí se generan ideas, conceptos y diseños para abordar el problema identificado. Se incluye la creación de prototipos y el desarrollo de soluciones tangibles que puedan ser implementadas.

Validación de la solución: Se evalúa y prueba la solución diseñada, recopilando retroalimentación de los usuarios y otras partes interesadas. Esto permite realizar ajustes y mejoras basados en la retroalimentación recibida, asegurando que el diseño final cumpla con los requisitos y expectativas establecidos. En *Design Science* existen diversas técnicas para la validación de la solución.

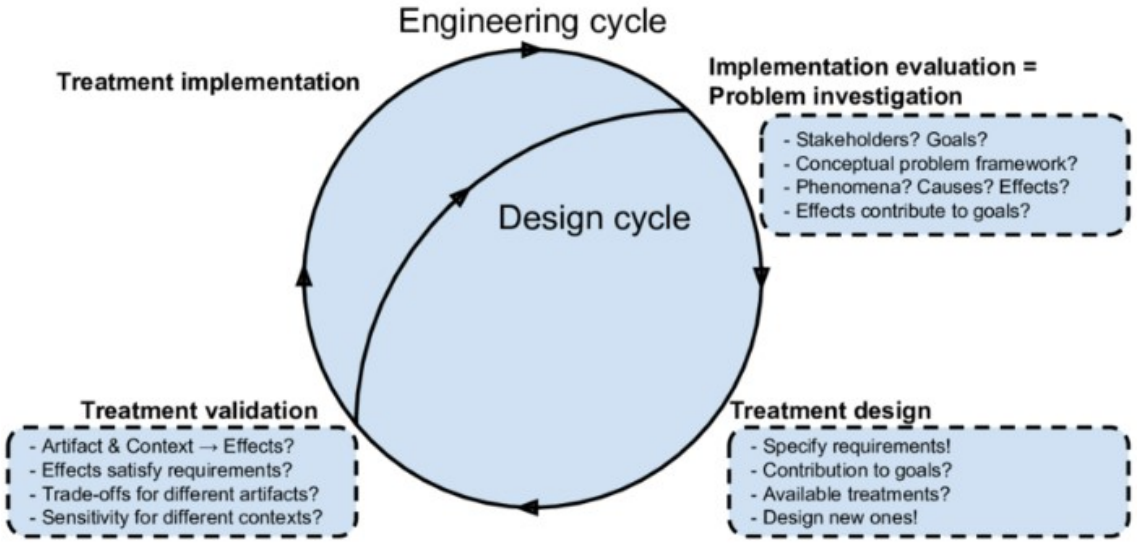

Figura 1.2: Fases del ciclo de diseño.

1.4 Estructura de la memoria

El presente Trabajo Fin de Grado se divide en diferentes capítulos, los cuales se encuadran con las fases del ciclo de diseño:

Capítulo 2: Investigación del Problema. Este capítulo se enfoca en comprender en profundidad el problema a abordar. Se detalla la estructura de datos que se manejará, se identifican y analizan los distintos problemas de seguridad que podrían surgir, y se explica la necesidad de desarrollar una arquitectura escalable. A partir de este análisis, se definen los requisitos específicos que la herramienta debe cumplir para ser efectiva

Capítulo 3: Diseño de la Solución. En este capítulo se describe detalladamente todos los componentes y microservicios que conforman la herramienta. Se proporcionan explicaciones sobre las tecnologías empleadas y se incluyen fragmentos relevantes de código. Además, se explica el proceso de despliegue de cada componente, especificando el tipo de servidor utilizado y las comunicaciones establecidas entre los distintos elementos del sistema.

Capítulo 4: Validación de la Solución. Este capítulo se dedica a validar la solución propuesta en el capítulo anterior. Para ello, se emplearán técnicas como los Experimentos de Mecanismo de Caso Único (*Single-Case Mechanism Experiments*). Estas técnicas permitirán evaluar la efectividad y eficiencia de la solución implementada.

Capítulo 5: Conclusiones. En el capítulo final, se presenta un informe detallado con las conclusiones obtenidas a lo largo del proyecto. Se resumen los hallazgos más relevantes, se discuten las implicaciones de los resultados y se ofrecen recomendaciones para trabajos futuros.

CAPÍTULO 2
Investigación del problema

Este capítulo tiene como objetivo responder a las cuatro primeras preguntas de investigación y proporcionar una visión detallada de los distintos aspectos del problema que serán abordados en el desarrollo del proyecto lo que implica aprender sobre el problema a tratar y estudiar su contexto. Se identifica qué aspectos necesitan ser mejorados y por qué [17]. Comprender a fondo el problema es fundamental antes de poder elicitar los requisitos necesarios para el diseño de la solución.

En primer lugar, es fundamental estudiar cómo se representan las variaciones genómicas, los problemas de seguridad que podrían surgir y la manera de obtener una arquitectura escalable capaz de soportar altos volúmenes de trabajo.

2.1 Formato de los datos

En el dominio de la genómica, dada la inmensidad y complejidad de los datos generados, se han desarrollado numerosos formatos de datos especializados para manejar diferentes tipos de información, desde secuencias de ADN hasta anotaciones genómicas y alineaciones. Sin embargo, cuando se trata de la representación y el intercambio de variantes genéticas, como SNPs, indels y otras mutaciones, se ha optado por utilizar el formato VCF (Variant Call Format) debido a su capacidad para manejar eficientemente estos datos. VCF se ha convertido en el estándar predominante en la comunidad científica, gracias a su flexibilidad, su estructura clara y su capacidad para incluir tanto la información sobre las variantes como los metadatos necesarios para interpretarlas correctamente.

El formato VCF es un formato de texto plano tabular que se organiza en líneas. Este formato es ampliamente utilizado en bioinformática para obtener información sobre variantes genéticas detectadas a partir de datos de secuenciación, las cuales pueden causar enfermedades, y por eso es de gran importancia poder representarlas con un formato estructurado para poder investigarlas y estudiarlas.

Las líneas de un archivo VCF se dividen en tres secciones principales: metadatos, cabecera y datos. Las líneas de metadatos comienzan con los símbolos '' y siguen un formato 'clave=valor'. Estas líneas proporcionan información adicional sobre el archivo VCF, como la versión de VCF utilizada, la fecha de creación, los criterios de filtrado aplicados durante el análisis, entre otros detalles relevantes. Además, proporcionan la estructura del fichero indicando que tipos de datos se representarán en las columnas INFO y FORMAT, ya que VCF es un formato flexible que acepta varios tipos de datos.

Después de las líneas de metadatos, se encuentra la línea de cabecera, que comienza con el símbolo 'CHROM'. Esta línea define los campos obligatorios y opcionales que describen las variantes presentes en el archivo. Cada campo tiene un nombre específico y se

representa en una columna de la línea de cabecera. Los campos adicionales definidos en los metadatos (que comienzan con ’’) se incluirán principalmente en la columna INFO, pero también pueden aparecer en las columnas de las muestras analizadas. La última columna (o columnas) de la línea de cabecera y las subsiguientes líneas de datos corresponden a estas muestras, donde se especifica la información genotípica y otros atributos para cada variante.

CHROM Esta columna representa el cromosoma o contig en el que se encuentra la variante. Puede contener el nombre del cromosoma o del contig, o un identificador numérico si los cromosomas están numerados.

POS Esta columna indica la posición específica dentro del cromosoma o contig donde se encuentra la variante. La posición se numera secuencialmente a lo largo del cromosoma o contig.

ID Esta columna contiene un identificador único para la variante. Puede ser un identificador de variante genérico, como un número, o un identificador específico si la variante ha sido previamente anotada en una base de datos genética.

REF Esta columna representa la base de referencia en la posición de la variante. Es la base que se encuentra en la secuencia de referencia del genoma en esa posición.

ALT Esta columna contiene la(s) base(s) alternativa(s) observada(s) en las muestras secuenciadas en comparación con la base de referencia. Puede contener una o varias bases alternativas, separadas por comas, que representan las variantes genéticas observadas.

QUAL Esta columna proporciona una medida de la calidad de la llamada de variantes. Es un valor numérico que indica la confiabilidad de la llamada del variante realizada por el programa de secuenciación o análisis.

FILTER Esta columna indica si la llamada de variantes ha pasado o no los criterios de filtrado aplicados durante el análisis. Puede contener el nombre de un filtro específico aplicado a la llamada del variante, como "PASS"si la llamada pasa todos los criterios de filtrado, o el nombre de un filtro específico si la llamada ha sido filtrada por ciertos criterios.

INFO Esta columna contiene información adicional sobre la variante en forma de campos de datos adicionales. Cada campo INFO proporciona detalles sobre la variante, como su frecuencia alélica, anotaciones funcionales, efectos en la proteína, entre otros.

FORMAT Describe el formato de los campos de datos que se encuentran en las columnas posteriores, que corresponden a las muestras individuales. Específicamente, indica qué tipos de información se incluyen para cada muestra y en qué orden se presentan.

Finalmente, después de la línea de cabecera, se encuentran las líneas de datos, donde cada línea representa una variación genética detectada en las muestras secuenciadas. [18]

```
1 ##fileformat=VCFv4.0
2 ##fileDate=20090805
3 ##source=myImputationProgramV3.1
4 ##reference=1000GenomesPilot-NCBI36
5 ##phasing=partial
```

```
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called genotypes">
##INFO=<ID=AC,Number=.,Type=Integer,Description="Allele count in genotypes, for each ALT allele, in the same order as listed">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=.,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
##ALT=<ID=DEL:ME:ALU,Description="Deletion of ALU element">
##ALT=<ID=CNV,Description="Copy number variable region">
#CHROM  POS ID  REF ALT QUAL  FILTER  INFO  FORMAT  NA00001  NA00002 NA00003
19  111 . A C 9.6 . . GT:HQ 0|0:10,10 0|0:10,10 0/1:3,3
19  112 . A G 10  . . GT:HQ 0|0:10,10 0|0:10,10 0/1:3,3
20  14370 rs6054257 G A 29  PASS  NS=3;DP=14;AF=0.5;DB;H2 GT:GQ:DP:HQ 0|0:48:1:51,51  1|0:48:8:51,51  1/1:43:5:.,.
20  17330 . T A 3 q10 NS=3;DP=11;AF=0.017 GT:GQ:DP:HQ 0|0:49:3:58,50  0|1:3:5:65,3  0/0:41:3:.,.
20  1110696 rs6040355 A G,T 67  PASS  NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 1|2:21:6:23,27  2|1:2:0:18,2  2/2:35:4:.,.
20  1230237 . T . 47  PASS  NS=3;DP=13;AA=T GT:GQ:DP:HQ 0|0:54:.:56,60  0|0:48:4:51,51  0/0:61:2:.,.
20  1234567 microsat1 G GA,GAC  50  PASS  NS=3;DP=9;AA=G;AN=6;AC=3,1  GT:GQ:DP  0/1:.:4 0/2:17:2  1/1:40:3
20  1235237 . T . . . . GT  0/0 0|0 ./.
X 10  rsTest  AC  A,ATG 10  PASS  . GT  0 0/1 0|2
```

Listing 2.1: Ejemplo de archivo VCF

2.2 Características y riesgos.

A la hora de desarrollar una solución para un problema de este estilo, se debe tener en cuenta ciertas características y riesgos relacionados con la seguridad y la privacidad de los datos. A continuación se muestra un listado de posibles riesgos de seguridad y privacidad.

1. **Violación de la Confidencialidad**: La **violación de la confidencialidad** ocurre cuando información sensible o privada es accedida por personas no autorizadas. Este tipo de riesgo compromete la capacidad de una organización o individuo para mantener secretos aquellos datos que no deben ser divulgados a terceros [19][20]

2. **Vulneración de la Intimidad**: La **vulneración de la intimidad** se refiere a la invasión o exposición indebida de la vida privada de una persona. En el contexto digital, este riesgo está relacionado con el acceso no autorizado, la recopilación, el uso o la divulgación de datos personales sin el consentimiento del individuo.[21][22]

3. **Corrupción o Destrucción de Datos**: La **corrupción o destrucción de datos** ocurre cuando la integridad de los datos es comprometida, ya sea por un ataque malicioso o por errores técnicos, lo que resulta en la modificación, pérdida o inaccesibilidad de la información.[23][24]

Para poder tratar dichos riesgos de una manera adecuada es necesario tener en cuenta los siguientes principios de seguridad e implementarlos junto a la solución.

1. **Gestión de Identidad**: La **gestión de identidad** se refiere al proceso de identificar y autenticar usuarios, sistemas o dispositivos que acceden a una red o sistema. Este principio es fundamental para asegurar que solo las personas o entidades autorizadas puedan acceder a información y recursos. La gestión de identidad ayuda a prevenir la *violación de la confidencialidad*, ya que asegura que los accesos no autorizados a datos sensibles sean evitados mediante un control riguroso de las identidades.

2. **Autorización**: La **autorización** es el proceso que define y controla los permisos que un usuario o sistema tiene sobre los recursos y datos dentro de un sistema. Este principio garantiza que solo los usuarios con los derechos adecuados puedan realizar acciones específicas sobre los recursos. La autorización está estrechamente relacionada con la *vulneración de la intimidad*, ya que un control adecuado sobre qué información puede ser accedida y por quién previene el acceso no autorizado a datos personales.

3. **Control de Acceso**: El **control de acceso** es un mecanismo de seguridad que regula quién puede interactuar con los sistemas y recursos, y de qué manera. Incluye tanto la autenticación como la autorización. Un control de acceso efectivo ayuda a mitigar riesgos como la *violación de la confidencialidad*, al asegurar que solo los usuarios autorizados tengan acceso a información crítica, y la *vulneración de la intimidad*, al restringir el acceso a datos personales sensibles.

4. **Gestión de la Privacidad**: La **gestión de la privacidad** implica proteger la información personal y asegurar que su recopilación, uso y divulgación cumpla con las políticas y leyes pertinentes. Este principio es crucial para prevenir la *vulneración de la intimidad*, ya que asegura que los datos personales se manejen de manera que se respete la privacidad del individuo, evitando usos indebidos o exposiciones no autorizadas.

5. **Registros de Auditoría**: Los **registros de auditoría** son registros detallados de las actividades y transacciones realizadas en un sistema. Estos registros permiten rastrear y revisar accesos, modificaciones y otros eventos críticos. La implementación de registros de auditoría ayuda a identificar y responder a incidentes de *corrupción o destrucción de datos*, al proporcionar una manera de rastrear los cambios en los datos y detectar cualquier actividad sospechosa o no autorizada.

6. **Cifrado**: El **cifrado** es el proceso de transformar datos en un formato codificado que solo puede ser descifrado por usuarios autorizados. Protege la información durante su almacenamiento y transmisión, asegurando que los datos sean ilegibles para personas no autorizadas. El cifrado es esencial para prevenir la *violación de la confidencialidad*, al proteger los datos de accesos no autorizados, y también contribuye a la *vulneración de la intimidad* al garantizar que la información personal esté segura durante su transmisión o almacenamiento.

2.3 Arquitectura

Otro aspecto a tener en cuenta es el tipo de arquitectura que se va a implementar. En el desarrollo de software, la **arquitectura monolítica** se refiere a un enfoque en el cual una aplicación es construida como una única unidad indivisible. En una arquitectura monolítica, todos los componentes funcionales de la aplicación como la interfaz de usuario, la lógica de negocio, y la gestión de datos están integrados en un solo código base. Esto significa que todas las funciones están interrelacionadas y son ejecutadas como un todo dentro de un único proceso.

La simplicidad en la organización del código y el despliegue es una de las principales ventajas de la arquitectura monolítica, lo que facilita su desarrollo inicial y su mantenimiento en etapas tempranas. Sin embargo, este enfoque también presenta desventajas significativas a medida que la aplicación crece en tamaño y complejidad. La naturaleza integrada de una aplicación monolítica puede dificultar la escalabilidad, ya que cualquier cambio en un componente puede requerir la recompilación y el rediseño de toda la aplicación. Además, la escalabilidad horizontal, que implica la distribución de la carga entre múltiples servidores, es limitada en arquitecturas monolíticas debido a la dependencia de una única instancia de aplicación.

Para superar estas limitaciones y lograr una mayor escalabilidad, las organizaciones han adoptado la **arquitectura de microservicios**. Este enfoque fragmenta la aplicación en componentes más pequeños y autónomos, conocidos como microservicios. Cada microservicio se encarga de una función específica y se ejecuta en su propio proceso, pudiendo ser desarrollado, desplegado, y escalado de manera independiente. La comunicación entre microservicios se realiza generalmente a través de interfaces bien definidas, como API (*Application Programming Interface*) basadas en protocolos ligeros como HTTP/REST o mensajería asíncrona.

La principal ventaja de la arquitectura de microservicios radica en su capacidad para mejorar la escalabilidad y la flexibilidad del desarrollo. Al descomponer una aplicación en microservicios, es posible escalar únicamente los componentes que requieren mayores recursos, sin necesidad de replicar la aplicación completa. Esto no solo optimiza el uso de recursos, sino que también facilita la implementación de mejoras y correcciones en partes específicas del sistema sin afectar a todo el conjunto.

Además, la arquitectura de microservicios promueve una organización más modular del equipo de desarrollo, donde distintos equipos pueden trabajar de manera autónoma en diferentes microservicios, utilizando tecnologías y lenguajes de programación

que mejor se adapten a las necesidades de cada componente. Esta autonomía reduce las dependencias y acelera los ciclos de desarrollo y despliegue, resultando en una mayor agilidad organizativa.

2.4 Estado actual

En el punto de partida de este proyecto, el microservicio de generación de reportes ya ha sido desarrollado y se encuentra en funcionamiento. Por lo tanto, el enfoque principal del trabajo consistirá en la implementación de los microservicios para la subida y descarga de archivos. En la figura 2.1 se presenta un diagrama que ilustra el funcionamiento del microservicio de generación de reportes y la interacción de sus componentes.

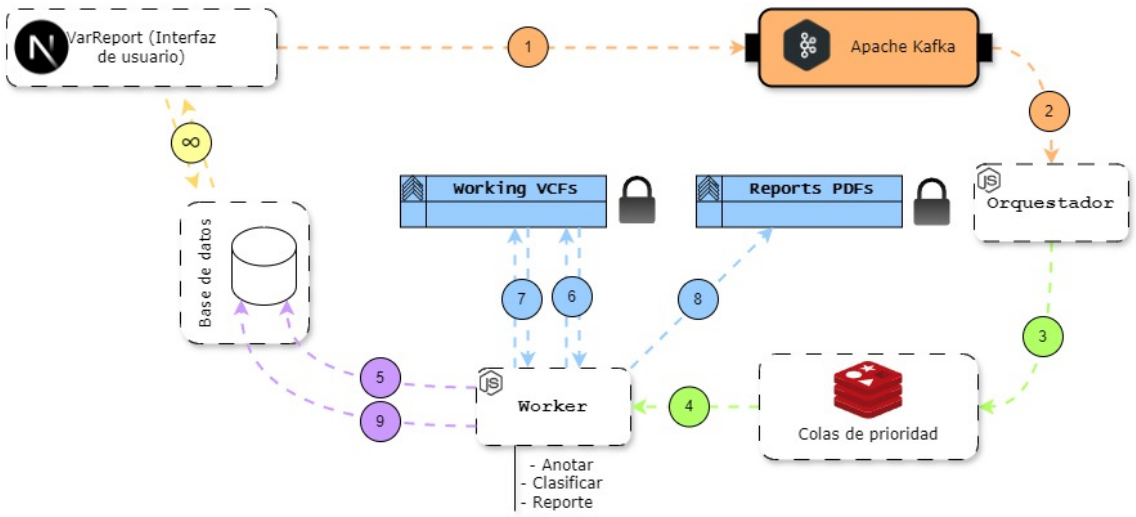

Figura 2.1: Diagrama del proceso de generación de reportes

El proceso de generación de reportes se compone de una serie de pasos que se describen a continuación a un nivel general. Es importante detallar este proceso, ya que algunos de sus componentes serán reutilizados en los microservicios de subida y descarga de archivos, y una comprensión clara de su funcionamiento es fundamental.

En primer lugar, desde la interfaz de usuario se produce un mensaje Kafka solicitando la generación de un reporte con los datos necesarios. Cabe destacar que, dado que la interfaz de usuario aún no está desarrollada, los mensajes deben ser generados manualmente para realizar las pruebas. Este mensaje es consumido por el orquestador, que se encarga de procesarlo, extraer la información relevante y añadir el trabajo a la cola de prioridad correspondiente.

El trabajador (worker) acepta el trabajo y ejecuta una serie de tareas. Primero, actualiza la base de datos asignando el estado *en ejecución* a la fila correspondiente al fichero en cuestión. Luego, realiza las anotaciones utilizando SnpEff, SnpSift y Python, como se muestra en la figura 2.2. Posteriormente, genera el archivo clasificado también utilizando Python y finalmente crea el reporte en formato PDF mediante la ejecución de un script de Python. Al concluir estas tareas, el estado del fichero en la base de datos se actualiza a *OK*.

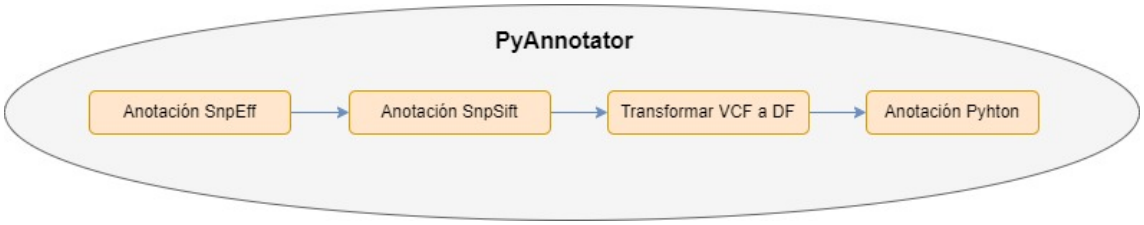

Figura 2.2: Proceso de anotación

Es importante destacar que tanto en este microservicio, como en todos los demás, se da solución y se abordan los principios de seguridad definidos en la sección 2.2. Los principios de **gestión de identidad**, **autorización**, **control de acceso** y **registros de auditoría**, se abordan con Auth0. Auth0 es una plataforma de identidad como servicio (IDaaS) que ofrece soluciones para autenticación, autorización y gestión de identidades para aplicaciones web, móviles y APIs. Es ampliamente utilizada para implementar sistemas de autenticación y autorización de manera segura y escalable, sin la necesidad de que las organizaciones construyan y mantengan su propia infraestructura de autenticación.

Para gestionar identidades, Auth0 ofrece un portal de autenticación, dónde los usuarios pueden iniciar sesión de diferentes maneras (contraseña, google, redes sociales...), además los administradores pueden crear y eliminar cuentas de usuario así como gestionar atributos y perfiles de usuario.

En cuanto a la autorización, Auth0 permite gestionarla mediante:

- **Roles y permisos**: Permite definir roles y asociarles permisos específicos. Los roles pueden asignarse a usuarios, lo que facilita el control granular sobre lo que cada usuario puede hacer dentro de una aplicación.
- **Reglas y políticas**: Auth0 permite la creación de reglas personalizadas que se ejecutan durante el proceso de autenticación, lo que permite implementar lógica de autorización específica, como restringir el acceso en función de atributos del usuario, dirección IP, ubicación geográfica, entre otros.
- **Tokens de acceso**: Auth0 utiliza tokens como JWT (JSON Web Tokens) para autorizar el acceso a recursos. Estos tokens incluyen información de autorización que el backend puede verificar sin necesidad de consultas adicionales.

Auth0 proporciona un control de acceso robusto al añadir una capa adicional de seguridad mediante la autenticación multifactor (MFA), que exige una segunda forma de verificación, como SMS o aplicaciones de autenticación, al iniciar sesión. Además, permite ajustar los controles de acceso en función del contexto del usuario, como la ubicación, el dispositivo o el comportamiento habitual, lo que contribuye a detectar y prevenir accesos no autorizados. También implementa políticas de seguridad como el bloqueo automático de cuentas después de múltiples intentos fallidos de inicio de sesión, para evitar ataques de fuerza bruta, y gestiona de manera segura las sesiones de usuario, permitiendo establecer políticas de expiración de sesión y reautenticación según sea necesario.

Existen dos niveles de registros de auditoría: El de Auth0 y uno interno a la herramienta. Auth0 ofrece capacidades de auditoría que incluyen el registro de eventos de seguridad, almacenando eventos detallados como inicios de sesión exitosos, fallidos, cambios en los perfiles de usuario, actualizaciones de permisos, entre otros. También es posible configurar notificaciones y alertas para eventos específicos, como intentos de acceso no autorizados, permitiendo a los administradores reaccionar rápidamente ante posibles incidentes de seguridad. La herramienta además, genera sus propios registros diarios y cada día se rotan.

La gestión de la privacidad dentro del sistema se realiza de manera interna, garantizando que cada usuario pueda interactuar únicamente con aquellos ficheros que le pertenecen o para los cuales ha recibido permisos explícitos. Aunque Auth0 no se encarga directamente de esta gestión específica de los permisos de acceso a los ficheros, juega un papel fundamental al proporcionar un mecanismo robusto de identificación y asignación de roles para cada usuario. Este enfoque asegura un control de acceso preciso y adecuado, fortaleciendo así la seguridad y privacidad de los datos manejados por el sistema.

Finalmente, todos los ficheros almacenados en el sistema, incluidos tanto los archivos PDF como los VCF, son cifrados internamente para garantizar la seguridad de los datos. Cuando un usuario solicita acceso a estos ficheros a través de la aplicación, los archivos se descifran de manera controlada antes de ser enviados al destinatario autorizado. Este mecanismo de cifrado y descifrado previene accesos no autorizados, ya que impide la lectura directa de los archivos desde su ubicación en el servidor, forzando que todas las interacciones con los ficheros se realicen a través de la aplicación, lo que refuerza significativamente la seguridad de la información.

2.4.1. Estructura de ficheros

Es necesario almacenar y estructurar los ficheros que se manejan. Los archivos se estructuran en tres subdirectorios principales, denominados bronce, plata y oro.

El nivel bronce se subdivide en dos subdirectorios: *initial vcfs* y *working vcfs*. En el subdirectorio *initial vcfs* se almacenan los archivos VCF (Variant Call Format) que son subidos por los usuarios. Estos archivos se guardan cifrados para garantizar la seguridad de los datos. Por otro lado, en el subdirectorio *working vcfs* se encuentran los archivos VCF que han sido comprimidos e indexados tras su carga inicial, facilitando así su acceso y procesamiento eficiente.

El nivel plata, de manera similar al nivel bronce, se divide en dos subdirectorios: *annotated vcfs* y *classified vcfs*. El subdirectorio *annotated vcfs* contiene archivos en formatos VCF y CSV que resultan de los diversos procesos de anotación aplicados sobre el VCF original, enriqueciendo así los datos con información adicional relevante. En el subdirectorio *classified vcfs* se almacena un archivo CSV que es el resultado de la clasificación del VCF original, incluyendo las categorías y etiquetas asignadas a las variaciones encontradas en el archivo original.

Finalmente, en el nivel oro se almacenan los informes en formato PDF generados a partir de los ficheros de los niveles bronce y plata. Estos informes están cifrados para asegurar la confidencialidad de la información y se encuentran listos para ser descargados por los usuarios.

En la figura 2.3 se muestra un esquema de dicha estructura.

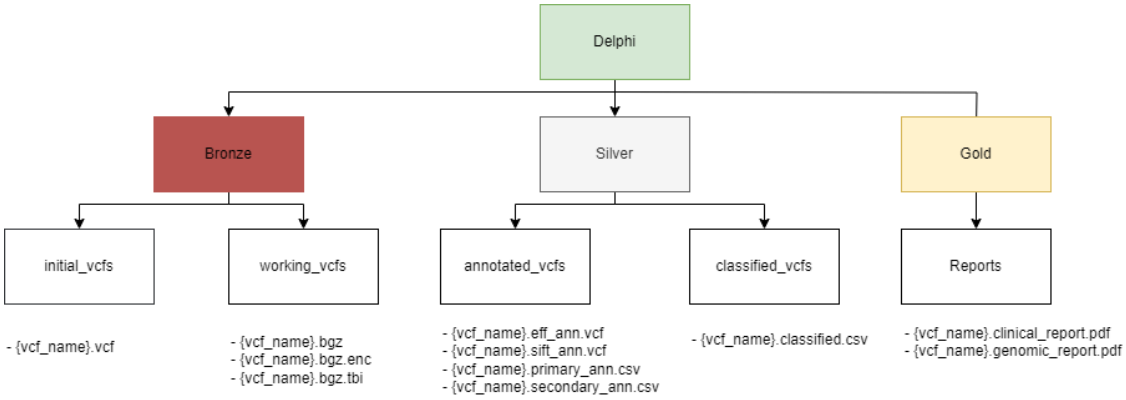

Figura 2.3: Organización de la estructura de ficheros.

2.5 Respuesta a preguntas de investigación

En las secciones anteriores se han abordado y estudiado las cuatro primeras preguntas de investigación. En esta sección se la dará respuesta a dichas preguntas teniendo en cuenta la información ya presentada.

¿Cuál es el formato de los datos genómicos típicos? El formato que se utilizará será el VCF. El formato VCF se destaca como una herramienta fundamental para trabajar con variantes genéticas debido a su capacidad para manejar grandes volúmenes de datos con eficiencia y claridad. Este formato de archivo es ampliamente aceptado en la investigación genética porque permite una representación compacta y estandarizada de variantes genómicas, como SNPs, indels y otras alteraciones estructurales. Además, el VCF incluye información detallada sobre la calidad de las variantes, los genotipos de los individuos y las anotaciones funcionales, lo que facilita la interpretación y comparación de los datos entre diferentes estudios y plataformas. Su estructura basada en texto plano y su compatibilidad con diversas herramientas bioinformáticas hacen del VCF una elección ideal para la integración y análisis de datos genéticos en diversas aplicaciones, desde la investigación básica hasta la medicina personalizada.

¿Cuáles son las características más relevantes a tener en cuenta al trabajar con datos genómicos? Las características de seguridad y privacidad son muy importantes a la hora de desarrollar una plataforma de este estilo, abordar dichas características es crucial debido a la sensibilidad inherente de la información genética, que puede revelar detalles íntimos sobre la salud, predisposiciones y características personales de los individuos. La información genética es única y, si se maneja de manera inapropiada, puede ser utilizada para discriminar o dañar a las personas, comprometiendo su privacidad y derechos. Además, el riesgo de ciberataques y accesos no autorizados puede poner en peligro datos extremadamente valiosos y personales. Implementar robustas medidas de seguridad y privacidad, como el cifrado de datos, controles de acceso estrictos y políticas claras de manejo de la información, es fundamental para proteger los datos genéticos de usos indebidos y garantizar la confianza de los usuarios en la aplicación. Esto no solo cumple con las regulaciones y estándares éticos, sino que también promueve la integridad y el respeto hacia la información personal de cada individuo.

¿Cuál es la arquitectura más adecuada? Finalmente, se ha optado por implementar una arquitectura de microservicios. Su uso ofrece varias ventajas significativas sobre

una arquitectura monolítica, especialmente en aplicaciones complejas y escalables. A diferencia de los sistemas monolíticos, donde todos los componentes están integrados en una única unidad, los microservicios dividen la aplicación en servicios independientes que se comunican entre sí a través de interfaces bien definidas. Esta separación permite un desarrollo más ágil, ya que los equipos pueden trabajar en diferentes servicios de manera autónoma y desplegar actualizaciones de manera independiente, reduciendo el riesgo de que un cambio en un componente afecte al sistema en su totalidad. Además, la arquitectura de microservicios facilita la escalabilidad, permitiendo que cada servicio se escale de forma independiente según la demanda, lo que optimiza el uso de recursos y mejora el rendimiento. También promueve una mayor resiliencia, ya que la falla de un servicio no necesariamente impacta a toda la aplicación, y ofrece flexibilidad para adoptar nuevas tecnologías y herramientas en servicios específicos sin afectar el conjunto del sistema. Estas características hacen que los microservicios sean una elección robusta y eficiente para aplicaciones que requieren alta disponibilidad, escalabilidad y agilidad en el desarrollo.

¿Cuál es el estado actual de la plataforma a mejorar? La plataforma se encuentra en una fase avanzada de desarrollo, en la cual se ha logrado implementar y poner en funcionamiento un microservicio con éxito. Asimismo, se ha llevado a cabo la selección de los componentes y tecnologías que constituirán la base de la plataforma. No obstante, a pesar de estos avances, aún persisten importantes tareas por completar. Es necesario desarrollar y poner en marcha un número adicional de microservicios, así como extender la utilización de los componentes seleccionados a lo largo de toda la plataforma. Estas actividades son esenciales para alcanzar la funcionalidad completa y el potencial deseado de la plataforma, garantizando así su eficacia y su capacidad para cumplir con los objetivos previstos.

CAPÍTULO 3
Diseño de la solución

En este capítulo se presentará en detalle la solución desarrollada, comenzando con una descripción del contexto en el que se lleva a cabo este desarrollo. Se abordarán los diversos recursos y componentes disponibles, así como su localización. A continuación, se explicará la estructura interna de los archivos utilizados y se describirá el estado actual del desarrollo.

Posteriormente, se ofrecerá una descripción exhaustiva de los distintos microservicios desarrollados, abarcando aspectos como su diseño, flujo de trabajo, detalles de implementación y los procedimientos necesarios para su despliegue.

3.1 Elicitación de requisitos

Elicitar los requisitos correctamente es clave para el éxito de cualquier proyecto. No definir y documentarlos conlleva a malentendidos con los stakeholdes, revisiones costantes y retrasos innecesarios. Estudios demuestran como una pobre elicitación de los requisitos pueden llegar a aumentar el coste y el tiempo en un 60 %. [25]

Existen dos tipos de requisitos, los funcionales y los no funcionales. Los requisitos funcionales, como su nombre indica, son las funcionalidades que debe tener el sistema, es decir, definen qué debe hacer el sistema, como por ejemplo, operaciones y flujos de trabajo que el sistema debe realizar, formatos y validaciones de entrada y salida de datos, comportamiento de la interfaz de usuario, requisitos de seguridad e integración de datos, etc.

Por otro lado, los requisitos no funcionales definien restricciones en el diseño del sistema y cómo debe ser construido (tecnologías utilizadas, tiempos de respuesta, estándares, compatibilidad...).[25] La lista de requisitos de este Trabajo Fin de Grado se expone en las tablas 3.1 y 3.2

Nombre	Descripción	Propiedad	Indicador
Requisito 1	Si el informe no se puede generar al momento, se añadirá a una cola y no se perderá.	Seguridad	Disponibilidad
Requisito 2	El usuario deberá ser capaz de saber el estado del análisis en tiempo real.	Seguridad	Disponibilidad
Requisito 3	El sistema deberá identificar el assembly del fichero automáticamente.	Usabilidad	Esfuerzo para usar
Requisito 4	El usuario podrá lanzar todos los análisis que quiera	Utilidad	Opinión de stakeholder
Requisito 5	El usuario podrá relanzar análisis sobre ficheros ya analizados.	Utilidad	Opinión de stakeholder
Requisito 6	El usuario debe poder subir ficheros vcf.	Utilidad	Opinión de stakeholder
Requisito 7	El usuario debe poder descargar informes en pdf.	Utilidad	Opinión de stakeholder
Requisito 8	el usuario debe poder eliminar ficheros vcf del sistema.	Utilidad	Opinión de stakeholder
Requisito 9	Si no se puede identificar el assembly sólo se le pida al usuario.	Usabilidad	Esfuerzo para usar

Tabla 3.1: Requisitos funcionales.

Nombre	Descripción	Propiedad	Indicador
Requisito 1	El informe se deberá generar en menos de una hora.	Eficiencia	Tiempo de respuesta
Requisito 2	La información en el servidor deberá cifrarse en todo momento.	Seguridad	Privacidad
Requisito 3	El usuario podrá visualizar sus análisis pero no los de otros.	Seguridad	Privacidad
Requisito 4	Se usará un sistema de auntenticación robusto	Seguridad	Privacidad
Requisito 5	Los ficheros deberán subirse en menos de 1 minuto y notificar al usuario.	Eficiencia	Tiempo de respuesta
Requisito 6	EL usuario podrá visualizar un análisis en menos de 5 segundos después de clicar	Eficiencia	Tiempo de Respuesta

Tabla 3.2: Requisitos no funcionales.

3.2 Contexto

En esta subsección se describen los diversos componentes que forman la arquitectura del sistema. Estos componentes se han dividido en dos subgrupos: los componentes que se encuentran en local (es decir, en un servidor propio) y los componentes que se encuentran en la nube. Los componentes que se encuentran en el servidor local son los siguientes:

Uploader y Downloader: Estos son los puntos de entrada a los microservicios de subida y descarga de ficheros. La interfaz de usuario se conecta a ellos mediante HTTPS, y posteriormente, envían solicitudes de trabajo al orquestador utilizando Apache Kafka.

Orquestador: El orquestador recibe solicitudes de trabajo del Uploader, el Downloader y la interfaz de usuario. Se encarga de asignar estas solicitudes a los workers utilizando colas de prioridad en Redis.

Workers: Los workers son responsables del trabajo computacional, incluyendo la compresión, indexación y anotación de archivos, así como la generación de informes en formato PDF.

Almacenamiento: Todos los archivos, tanto los archivos VCF como los informes PDF, se almacenan en el sistema de archivos del servidor local.

Los componentes que se encuentran en la nube son los siguientes:

Apache Kafka: Es una plataforma de transmisión de eventos utilizada para comunicar los puntos de entrada con el orquestador.

Redis: Redis se utiliza para gestionar las colas de prioridad que permiten la asignación eficiente de tareas a los workers.

Base de datos: La base de datos guarda metadatos sobre los archivos VCF y otra información adicional, como el estado de los informes PDF asociados a los archivos VCF. No almacena el contenido de los archivos.

Interfaz de usuario: La interfaz de usuario se desplegará utilizando Vercel, proporcionando una plataforma accesible y eficiente para la interacción con el sistema.

La arquitectura del sistema se compone de una serie de componentes distribuidos entre un servidor local y la nube. En el servidor local, se gestionan las operaciones críticas de subida, descarga, procesamiento y almacenamiento de archivos. En la nube, se utilizan herramientas avanzadas como Apache Kafka y Redis para la comunicación y la gestión eficiente de tareas, así como una base de datos para el almacenamiento de metadatos y una interfaz de usuario desplegada en Vercel para facilitar la interacción del usuario con el sistema. Esta combinación de componentes locales y en la nube permite una arquitectura robusta, escalable y eficiente.

3.3 Subida de archivos

En esta sección se hablará sobre el microservicio de subida de archivos. Se utilizará para que el usuario pueda subir sus archivos vcf al servidor para en un futuro generar el informe correspondiente.

3.3.1. Diseño

En este microservicio intervendrán muchos componentes que fueron utilizados en la generación de informes (colas de trabajo, envío de mensajes, base de datos, etc). Además intervendrá un nuevo componente clave, el uploader. El diagrama de flujo correspondiente se muestra en la figura 3.1.

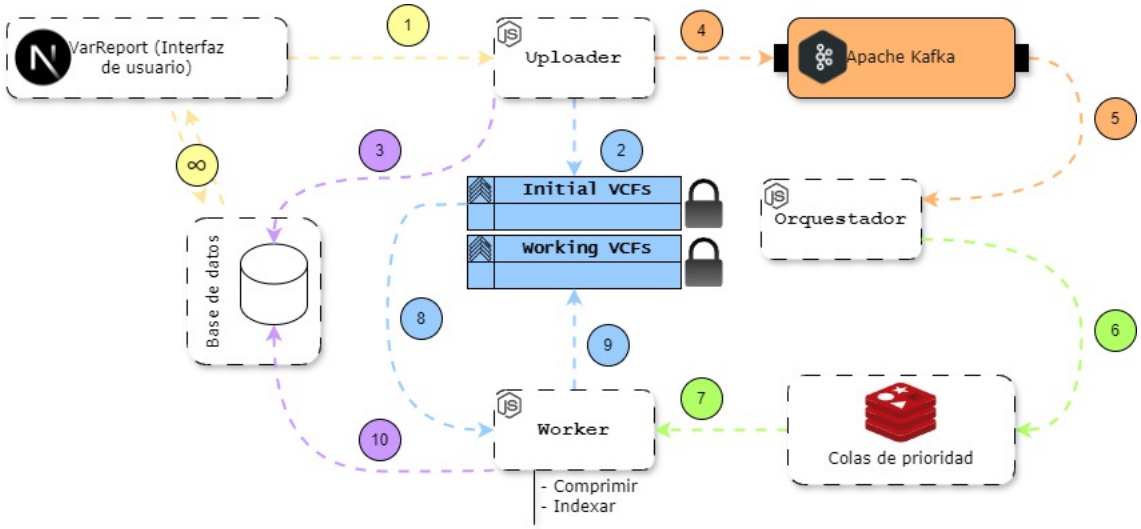

Figura 3.1: Diagrama del proceso de subida de archivos

Desde la aplicación se manda una petición HTTPS con el contenido del archivo a subir que llega al uploader, éste almacena el archivo en el disco local y crea una entrada en la base de datos con los datos del archivo, posteriormente, produce un mensaje kafka solicitando la compresión y la indexación del archivo. El orquestador consume dicho mensaje y añade la tarea a la cola correspondiente, un worker aceptará la tarea en la cual leerá el archivo almacenado, generará los archivos indexados y comprimidos y actualizará la base de datos asignando el estado del archivo como *no report*.

3.3.2. Implementación

En este subapartado se explicará detalladamente el funcionamiento de cada componente, las tecnologías utilizadas y se presentarán fragmentos de código relevantes.

Se ha decidido utilizar JavaScript como lenguaje de programación principal. La elección de JavaScript y Node.js se fundamenta en varias ventajas significativas. En primer lugar, su alta eficiencia en operaciones de entrada/salida (I/O) y su naturaleza asíncrona facilitan la integración con diferentes componentes del sistema. Además, el amplio ecosistema de npm proporciona una vasta cantidad de paquetes y librerías que aceleran y simplifican el proceso de desarrollo.

Node.js permite unificar el desarrollo del frontend y del backend utilizando el mismo lenguaje, lo que promueve una mayor coherencia en el código y mejora la productividad del equipo de desarrollo. La robustez de Node.js y su activa comunidad aseguran un soporte continuo, así como la disponibilidad de recursos y actualizaciones constantes, lo que resulta fundamental para mantener el proyecto actualizado y funcional a lo largo del tiempo.

El componente de carga de archivos debe ser capaz de recibir peticiones HTTPS para almacenar los archivos, por lo que el primer paso consiste en configurar un servidor

HTTPS. Para este propósito, se ha optado por utilizar Express.js debido a su simplicidad para la creación de servidores web. Es importante destacar que es necesario disponer de certificados válidos para poder lanzar el servidor con HTTPS, como se ilustra en el código 3.1.

```
const server = process.env.NODE_ENV === "production" ? https.
    createServer({
  cert: fs.readFileSync(process.env.CERT_PATH),
  key: fs.readFileSync(process.env.KEY_PATH),
}, app) : http.createServer(app);
```

Listing 3.1: Creación del servidor HTTPS

Una vez el servidor está operativo, es necesario configurar la ruta a la cual se enviarán las peticiones y definir el método HTTP que se utilizará. Esto se puede realizar fácilmente en Express mediante la función `app.post` (existen otras funciones como `app.get` o `app.put`, pero para subir información al servidor se ha decidido utilizar el método POST), especificando la ruta como parámetro.

Es necesario validar la petición, para ello, la petición debe incluir una cabecera `code` con el código del archivo (por ejemplo, cardio, onco, etc.). En caso de que no se incluya esta cabecera, se enviará una respuesta de error con el código 400.

Se ha decidido utilizar Auth0 para añadir autenticación y seguridad. Las peticiones deben incluir además una cabecera `Authentication` con un JWT (JSON Web Token) válido. Si esta cabecera no está presente o el token no es válido, se enviará una respuesta de error con el código de estado 401 (Unauthorized). Para implementar esta funcionalidad se utiliza la biblioteca `express-oauth2-jwt-bearer` de Auth0, como se muestra en la código 3.2. Los valores de los parámetros necesarios se deben obtener creando una aplicación en la web de Auth0.

```
const { auth } = require("express-oauth2-jwt-bearer");

require("dotenv").config();

const checkJwt = auth({
  audience: process.env.AUDIENCE,
  issuerBaseURL: process.env.ISSUER,
  tokenSigningAlg: process.env.SIGNINGALG,
});

module.exports = checkJwt;
```

Listing 3.2: Uso de la biblioteca de auth0 para validar tokens

Una vez la petición es validada, se deben almacenar los ficheros subidos en disco. Se ha decidido utilizar la biblioteca *multer*, que permite leer el contenido de un archivo en la petición y guardarlo en el espacio de disco configurado (ver código 3.3). Es importante destacar que *multer* actúa como middleware, por lo que se deberá pasar una función de *multer* como parámetro en la función de recepción de la petición. Además, para que *multer* pueda leer el contenido correctamente, los datos en la petición deben enviarse como *form-data*, donde la clave se especifica en el código (ver código 3.4). En este caso, la clave asignada es "files", por lo que en la petición se deben enviar los datos en formato *form-data* donde "files.[es] la clave y el contenido del archivo es el valor. Con la función `upload.array` se indica que puede haber más de un fichero en la petición, y cada fichero deberá usar la misma clave.

```
const storage = multer.diskStorage({
  destination: function (req, files, callback) {
    const savePath = process.env.INITIAL_VCFS;
    const savePathWithCode = path.join(savePath, req.headers.
       code);

    try {
      verifyFilePath(savePathWithCode);
      callback(null, savePathWithCode);
    } catch (error) {
      callback(error, null);
    }
  },
  filename: function (req, file, callback) {
    callback(null, file.originalname);
  },
});
```

Listing 3.3: Configuración de multer y destino de los archivos

```
vcf_router.post(
  "/upload_vcfs",
  upload.array("files"),
```

Listing 3.4: Uso de multer como middleware en la recepción de la petición.

El *uploader* creará una conexión a la base de datos (código 3.5) y creará una entrada para cada fichero subido. Se ha utilizado la biblioteca de mariadb para JS ya que el SGBD que se usa es mariadb.

```
const pool = mariadb.createPool({
  database: process.env.DATABASE,
  host: process.env.DB_HOST,
  password: process.env.DB_PASS,
  user: process.env.DB_USER,
});

pool.getConnection().then((conn) => {
  conn.release().catch(console.error);
});

module.exports = pool;
```

Listing 3.5: Conexión a la base de datos

Los archivos subidos deben ser comprimidos e indexados, se enviará un mensaje mediante Apache Kafka al orquestador solicitando dicha tarea. Mediante el uso de la biblioteca de Kafka para JavaScript, se crea un cliente de Kafka, se configura como productor, se conecta al broker y se crea una función para mandar mensajes. Se destaca el método `getInstance()` (ver código 3.6). Esta función se encarga de devolver una instancia de Kafka, de forma que, al querer enviar un nuevo mensaje, se invoque dicho método y siempre devuelva la misma instancia en vez de crear una nueva cada vez, permitiendo ahorrar recursos. Una vez obtenida la instancia, se envía el mensaje en formato JSON con

un campo que indica el nombre de la cola (*CompressVcfFile*) y usando el tópico *generate_report* (ver código 3.7).

```
static getInstance() {
    if (!instance) {
      logger.info({
        label: "[kafkaClientService.js > getInstance]",
        message: "Kafka instance created successfully",
      });
      instance = new KafkaClientService();
    }
    logger.info({
      label: "[kafkaClientService.js > getInstance]",
      message: "Kafka instance retrieved successfully",
    });
    return instance;
  }
```

Listing 3.6: Método que devuelve la instancia

```
    const kafkaClientInstance = getInstance();

    const messages = info.files.map((file) => {
      return {
        value: JSON.stringify({
          queue: "compressVcfFile",
          code: file.code,
          fileName: file.filename,
        }),
      };
    });

    await kafkaClientInstance.sendMessages("generate_report",
       messages);
```

Listing 3.7: Envío del mensaje Kafka

El orquestador debe consumir el mensaje enviado, por lo que será necesario configurar un cliente de Kafka también. La estructura será similar a la del uploader, incluyendo el método `getInstance()`, pero configurando el cliente como consumidor en vez de productor (ver código 3.8). La instancia tendrá además un método `run()` que se ejecutará cada vez que se consume un mensaje. Este método recibe como parámetros el tópico (debe ser el mismo que el del productor) y una función que se ejecutará cada vez que se recibe un mensaje (ver código 3.9). Como se ha mencionado anteriormente, el mensaje está en formato JSON, donde uno de los campos es la cola a la que se debe añadir la tarea. Por lo tanto, esta función se encarga de parsear el mensaje, extraer el nombre de la cola y añadir la tarea (ver código 3.10).

```
class KafkaClientService {
  #kafka;
  #consumer;
```

Listing 3.8: Configuración de Kafka como consumidor

```
async run(topic, processMessage) {
    await this.#consumer.connect();
    await this.#consumer.subscribe({ topic, fromBeginning:
       false });
    await this.#consumer.run({
      eachMessage: processMessage,
    });
  }
```

Listing 3.9: Método run

```
const processMessage = async ({ message }) => {
  logger.info({
    label: "[orchestrator.js] > processMessage",
    message: `Request received: ${message.value.toString()}`,
  });

  const request = JSON.parse(message.value.toString());

  await queues[request.queue].add("newRequest", request);
};
```

Listing 3.10: Extracción de datos y encolamiento

Para implementar las colas de trabajo se ha usado la biblioteca BullMQ que proporciona colas de trabajo basadas en Redis. La clase Queue permite la fácil creación de colas de trabajos, como se puede observar en el código 3.11.

```
const { Queue, QueueEvents } = require("bullmq");
const queueNames = [
  "deleteVcf",
  "generateReport",
  "setAssembly",
  "compressVcfFile",
];

require("dotenv").config();

const connection = {
  host: process.env.REDIS_HOST,
  port: process.env.REDIS_PORT,
  user: process.env.REDIS_USER,
  password: process.env.REDIS_PASSWORD,
};

const queues = queueNames.reduce(
  (queues, queueName) => {
    queues[queueName] = new Queue(queueName, {
      connection,
    });
```

Listing 3.11: Creación de colas de trabajo

Un worker debe aceptar la tarea, la biblioteca BullMQ proporciona una clase Worker que permite la implementación de trabajadores. Estos trabajadores se puedes asociar fácilmente a una cola y recibir tareas de ahí (ver código 3.12).

```
const worker = new Worker(
  "compressVcfFile",
```

Listing 3.12: Creación de un worker asociandolo a una cola

Una vez el worker acepte la tarea, éste deberá leer el archivo subido y generar los archivos indexados y comprimidos, esto se realiza mediante la ejecución del siguiente comando de terminal.

```
bgzip -c ${inputFilePath} > ${compressedFilePath} && tabix -p
   vcf -f ${compressedFilePath}
```

La ejecución de comandos se realiza con el módulo *child process*. Por último, el worker realiza una conexión a la base de datos y actualiza el estado del fichero a *no report*.

3.3.3. Despliegue

El *uploader* y el *orchestrator* se ejecutan en un servidor local, por lo que es necesario desplegarlos. Los componentes que se ejecutan en la nube ya se encuentran desplegados.

Primero, se deben clonar los repositorios en el servidor mediante el comando:

```
git clone 'urlalrepositorio'
```

El paquete npm permite el fácil despliegue de aplicaciones Node.js. Con el comando:

```
npm install
```

se instalan todas las dependencias y paquetes necesarios. Luego, ejecutando:

```
npm start
```

el servicio se desplegará correctamente en el puerto indicado como variable de entorno, sin necesidad de realizar ninguna acción adicional. Sin embargo, esto presenta una desventaja significativa: al ejecutar `npm start`, no se recupera el control de la terminal, lo que requiere lanzar otra instancia para ejecutar más componentes. Además, el proceso es hijo de la terminal, por lo que si esta se cierra, el servicio se detendrá y tendrá que ser desplegado nuevamente.

Para solucionar estos inconvenientes, se ha optado por utilizar `pm2`. `pm2` es un gestor de procesos para Node.js que permite desplegar aplicaciones en segundo plano de forma independiente, sin que queden ligadas a la terminal y devolviendo el control. Además, `pm2` ofrece la capacidad de gestionar las distintas aplicaciones mediante comandos para detener, reiniciar y eliminar los servicios. También proporciona funcionalidades adicionales y servicios de monitoreo, como la visualización de los registros (*logs*) de las distintas aplicaciones en tiempo real.

El proceso de despliegue con `pm2` es también extremadamente sencillo. Simplemente con el comando:

```
pm2 start npm -- start
```

se desplegará la aplicación de forma correcta. Entre las ventajas adicionales de `pm2` se incluyen:

- **Persistencia de los procesos:** Los procesos gestionados por `pm2` pueden configurarse para reiniciarse automáticamente tras un fallo o después de reiniciar el servidor, asegurando así la disponibilidad continua del servicio.
- **Balanceo de carga:** `pm2` puede ejecutar múltiples instancias de una aplicación, distribuyendo la carga de trabajo entre ellas para mejorar el rendimiento y la fiabilidad.
- **Monitoreo detallado:** Ofrece una interfaz web y comandos de línea para monitorear el estado y rendimiento de las aplicaciones, incluyendo métricas como CPU y uso de memoria.
- **Facilidad de configuración:** La configuración de `pm2` se puede gestionar a través de archivos de configuración JSON o YAML, lo que permite un control preciso sobre el comportamiento de despliegue.

Estas características hacen de `pm2` una herramienta robusta y eficiente para el despliegue y gestión de aplicaciones Node.js en entornos de producción.

3.4 Descarga de informes

En esta sección se describirá el microservicio de descarga de informes. Se encargará de enviar al usuario los informes en formato PDF previamente generados.

3.4.1. Diseño

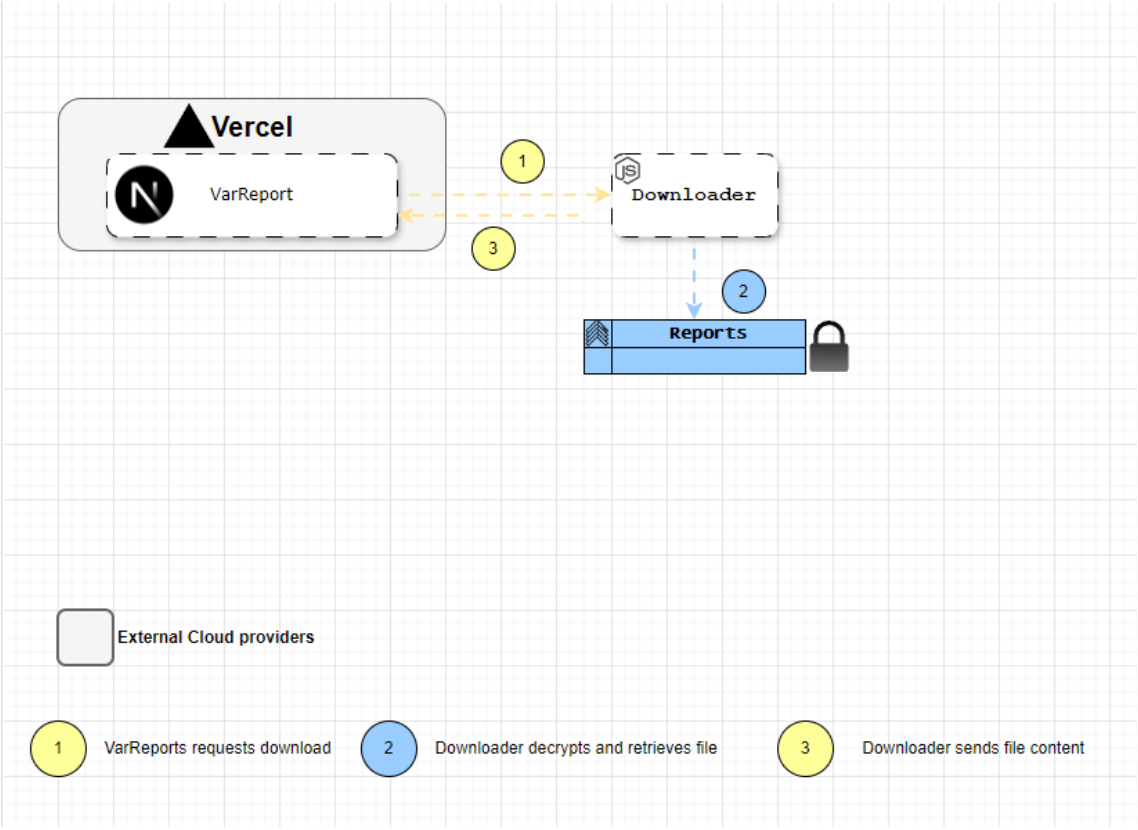

Figura 3.2: Diagrama del proceso de descarga de archivos.

Como se puede observar en el diagrama 3.2, el proceso de descarga de archivos es mucho más sencillo que el de subida, se empieza con una petición desde la interfaz de usuario que llega al *downloader*, éste busca el fichero solicitado, lo descifra y lo envía al usuario. Cabe recalcar que al no necesitar modificar la base de datos y el proceso de descifrado y envío no es costoso se ha optado que el propio *downloader* actúe como trabajador haciendo así que otros componentes como el orquestador, las colas de prioridad o Apache Kafka no sean necesarios, lo que conlleva a una menor carga de trabajo para dichos componentes y un mayor aprovechamiento de su capacidad para otras tareas más costosas, como por ejemplo, la generación de dichos reportes.

3.4.2. Implementación

Node.js y Express.js también serán las tecnolgías escogidas para el desarrollo. Al igual que el *uploader*, se desplegará un servidor HTTPS para recibir las peticiones del usuario y se usará auth0 para la validación de dichas peticiones. Las peticiones deberá incluir una cabecera *code* con el código del archivo, una cabecera *file_name* con el nombre del archivo VCF del que se quiere obtener el reporte PDF (el nombre del archivo PDF es desconocido por el usuario), otra cabecera *type* que indicará si se quiere descargar el informe clínico o genómico y por último una cabecera *mode* que indicará si se quiere descargar el archivo o sólamente visualizarlo en el navegador sin llegar a guardarlo en disco (ver código 3.13).

```
router.get("/report", (req, res, next) => {
  const mode = req.headers["mode"];
  if (
    !req.headers["code"] ||
    !req.headers["file_name"] ||
    !mode ||
    !req.headers["type"] ||
    !(mode === "view" || mode === "download")
  ) {
    res.status(400).send(missingHeaders);
    return;
  }
```

Listing 3.13: Comprobación de cabeceras en la petición.

El archivo debe ser descifrado, se usa un algoritmo estándar y una clave secreta para obtener el contenido. El contenido del archivo se envía al usuario a través de una respuesta HTTPS , en el caso que se haya especificado el modo descarga será necesario añadir una cabecera *Content-disposition* con un valor específico que se puede observar en la figura 3.14. Esto hará que el navegador guarde el contenido recibido en disco con el nobre de fichero especificado en la cabecera.

```
const sendContents = (data) => {
    if (mode === "view") {
      res.contentType("application/pdf");
      res.send(data);
    } else {
      const fileContents = Buffer.from(data, process.env.MODE)
          ;
      const readStream = new stream.PassThrough();
      readStream.end(fileContents);
      res.contentType("application/pdf");
      res.set(
```

```
        "Content-disposition",
        `attachment; filename=${path.basename(file_path)}`,
      );
      readStream.pipe(res);
    }
  };
```

Listing 3.14: Función de envío de contenido

3.4.3. Despliegue

El despliegue se realizará de forma idéntica al del apartado 3.3.3, se debe clonar el repositorio en el servidor, configurar un puerto que no esté en uso y ejecutar el proyeto con pm2.

3.5 Interfaz de Usuario

El componente final a desarrollar es la interfaz de usuario, a través de la cual los usuarios interactuarán para subir y descargar archivos, así como para generar informes. El objetivo primordial es diseñar una interfaz que sea sencilla e intuitiva, minimizando el esfuerzo requerido por parte del usuario para aprender a utilizarla. Se busca asegurar una experiencia de usuario fluida y accesible, facilitando todas las operaciones de manera eficiente y comprensible.

3.5.1. Diseño

En cuanto a su diseño, se ha decidido crear tres pantallas distintas: una pantalla de inicio de sesión, la pantalla principal y una pantalla de subida de archivos.

La pantalla de inicio (figura 3.3) es la primera que aparece al acceder a la URL. Esta pantalla proporciona una breve descripción de la herramienta y presenta un botón de inicio de sesión. Al pulsar este botón, se despliega un formulario que solicita las credenciales necesarias para la identificación y el inicio de sesión del usuario. El diseño de esta pantalla busca ser claro y directo, facilitando así el acceso y uso de la herramienta desde el primer momento.

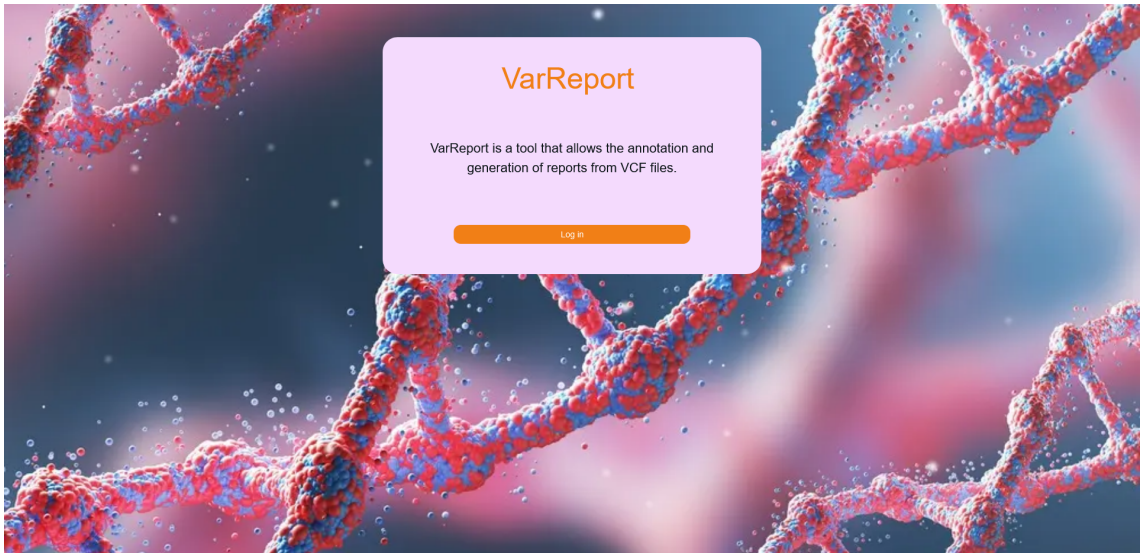

Figura 3.3: Pantalla de inicio de sesión.

Al iniciar sesión con éxito, el usuario será redirigido a la pantalla principal (figura 3.4). Esta pantalla contiene una tabla con una fila por cada archivo VCF disponible. La tabla proporciona información relevante sobre cada archivo, como su nombre, fecha de subida, última actualización y *assembly*. Sin embargo, lo más destacable son las acciones que se pueden realizar sobre cada archivo.

La tabla incluye dos campos, *summary report* y *detailed report*, que permiten realizar acciones sobre los informes generados. El botón con un símbolo de un ojo permite visualizar el informe, mientras que el botón con un símbolo de una flecha hacia abajo permite descargarlo. Es importante destacar que estos botones solo serán visibles si los informes ya han sido generados previamente; de lo contrario, solo se verá un espacio en blanco.

El último campo, *actions*, contiene dos botones adicionales: el primero permite eliminar el archivo, mientras que el segundo genera los reportes.

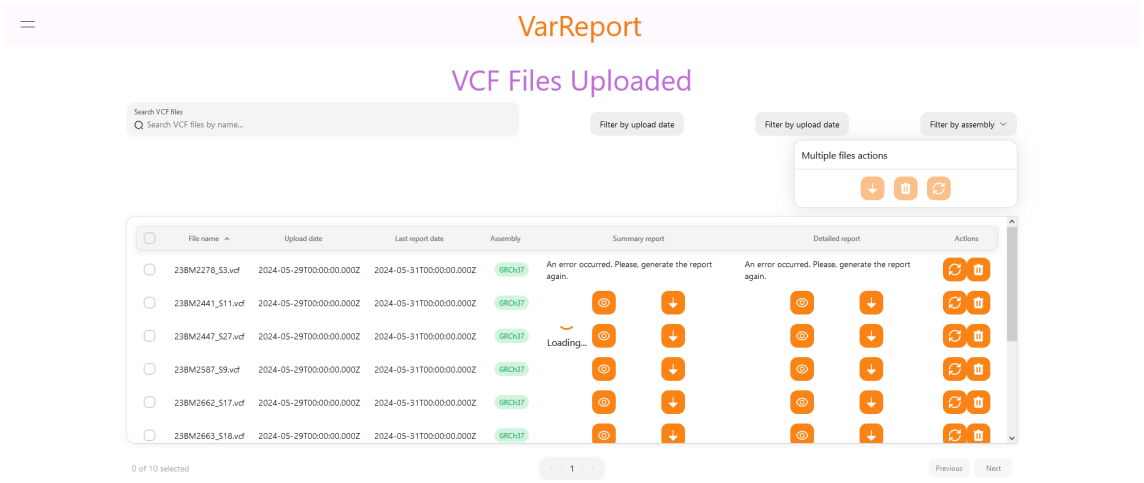

Figura 3.4: Pantalla principal.

También se ofrece la posibilidad de realizar acciones sobre más de un archivo a la vez, el usuario puede seleccionar varios archivos pulsando sobre la caja que se encuentra

a la izquierda del nombre del archivo, lo que habilitará la pestaña *Multiple file actions* que se encuentra arriba a la derecha de la tabla principal. Esta pestaña contiene tres botones similares a los de la tabla principal que permiten descargar los informes de todos los archivos seleccionados, eliminar todos los archivos seleccionados o generar los informes de todos los archivos seleccionados. Existen funcionalidades extra como el filtrado de archivos por nombre, por *assembly* o por fecha, navegación entre páginas, etc.

La pantalla de subida de archivos (figura 3.5) permite al usuario subir los archivos VCF que desee, contiene un cuadro de texto en el que se pueden arrastrar archivos para subirlos, además, si el usuario no desea arrastrar los archivos, también hay un botón *select files* a la derecha, que al pulsar sobre él, se abrirá el Gestor de Archivos de el sistema para poder buscar y seleccionar los archivos deseados.

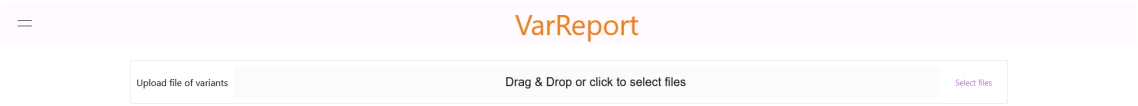

Figura 3.5: Pantalla de subida de archivos.

3.5.2. Implementación

Para implementar la interfaz de usuario se ha decido seguir optando por JavaScript como lenguaje de programación, además se usará el *framework* Next.js, Next.js es un *framewor* de desarrollo web basado en React que facilita la creación de aplicaciones web modernas, rápidas y optimizadas. Desarrollado por Vercel, Next.js amplía las capacidades de React al proporcionar herramientas y funcionalidades adicionales, mejorando el rendimiento, la experiencia del desarrollador y la capacidad de escalar proyectos. Entre sus características destacadas se encuentran el renderizado del lado del servidor (SSR) y la generación de sitios estáticos (SSG), lo que permite que las páginas se rendericen dinámicamente o se generen como HTML estático durante el proceso de build.

La estructura de rutas basada en archivos y el pre-renderizado automático simplifican el desarrollo, permitiendo a los desarrolladores centrarse en la lógica de la aplicación sin preocuparse por configuraciones complejas.

El rendimiento es otra área donde Next.js sobresale. Las optimizaciones automáticas, como el código dividido y la carga diferida, aseguran que las aplicaciones se carguen rápidamente y funcionen sin problemas. Esto se traduce en una mejor experiencia de usuario, con tiempos de carga más rápidos y una navegación más fluida. La facilidad de integración con API mediante las API Routes también hace que Next.js sea una excelente opción para aplicaciones full-stack, permitiendo la creación de endpoints API dentro de la misma aplicación.

La primera página a desarrollar es la página de inicio de sesión. Esta página es fundamental dentro del sistema, y su diseño es relativamente sencillo en comparación con otros componentes. Su estructura básica incluye una imagen de fondo que proporciona contexto visual, un título que introduce al usuario, un párrafo de texto explicativo y un botón de acción.

Para la autenticación de usuarios, al igual que en los microservicios previos, se ha optado por utilizar *Auth0*. La biblioteca de *Auth0* para *Next.js* facilita esta tarea al incorporar un panel de autenticación predeterminado, que permite tanto la identificación de usuarios como la creación de sesiones de manera eficiente. Esto elimina la necesidad de desarrollar un panel de autenticación personalizado, optimizando el proceso de autenticación.

En el código 3.15 se observa que el botón presente en la página de inicio de sesión está vinculado a una ruta específica. Al interactuar con este botón, el usuario es redirigido automáticamente al panel de autenticación predeterminado proporcionado por *Auth0*, simplificando así la experiencia de acceso al sistema.

```
<Button
as={Link}
className={styles.login_button}
href={"/api/auth/login"}
variant="flat"
>
Log in
</Button>
```

Listing 3.15: Implementación del botón de login.

La siguiente página a desarrollar es la página de inicio (*home*). Su contenido principal consiste en una tabla que muestra información relevante y proporciona acciones relacionadas con los ficheros VCF almacenados en el sistema.

Para obtener y mostrar los datos de los ficheros VCF, se utiliza la función `useSWR`. Esta función es un *hook* de *React* que recibe como parámetros una URL y una función *fetcher*. El *hook* `useSWR` ejecuta la función *fetcher*, pasando la URL como argumento, y espera recibir los datos correspondientes. La función *fetcher* se encarga de realizar una petición GET y devolver los datos obtenidos de la respuesta.

El *hook* `useSWR` retorna un objeto JSON, en el cual la propiedad `data` contiene los datos recuperados. La decisión de utilizar `useSWR` en lugar de invocar directamente la función *fetcher* se fundamenta en la capacidad de `useSWR` para ejecutar la función *fetcher* de manera periódica, garantizando así que los datos presentados en la tabla se mantengan siempre actualizados.

```
const { data, error, isLoading, isValidating } = useSWR(
    "/api/vcf/get_all_vcfs",
    fetcher,
    { fallbackData: [] },
  );
```

Listing 3.16: Función useSWR.

En el código 3.16 se ilustra el uso de la función `useSWR`. En este caso particular, se realiza una petición a una URL local que corresponde a un fichero `.js`. Este fichero establece una conexión con la base de datos, ejecuta una operación `SELECT`, y posteriormente *parsea* el resultado en un objeto JSON antes de devolverlo para su procesamiento posterior.

Antes de proceder con el renderizado de los datos obtenidos, es necesario definir ciertas variables clave. Para este propósito, se utiliza la función `useState` de *React*. Esta función recibe un valor inicial como argumento y devuelve una pareja compuesta por dicho valor y una función de actualización. El valor devuelto puede ser almacenado en una variable, y la función de actualización permite modificar el valor de dicha variable. Al invocar esta función con un nuevo valor como argumento, se actualiza el estado de la variable y se provoca un re-renderizado automático de la página, reflejando los cambios en la interfaz de usuario. En el código 3.17 se muestra un ejemplo de variables definidas utilizando `useState`.

```
1  // The keys of the selected rows.
2  const [selectedKeys, setSelectedKeys] = useState(new Set([])
       );
3
4  // The string that is used to filter the table by file name
5  const [filterFileName, setFilterFileName] = useState("");
6  // The dates that are used to filter the table by upload
       date
7  const [filterUploadDate, setFilterUploadDate] = useState(
       null);
8  // The dates that are used to filter the table by last
       report date
9  const [filterLastReportDate, setFilterLastReportDate] =
       useState(null);
10 // The list of assemblies that are selected. It is used to
       filter by assembly
11 const [assemblyFilter, setAssemblyFilter] = useState("all");
```

Listing 3.17: Uso de useState para definir variables.

En la variable *data* se almacenan todos los datos correspondientes a los ficheros VCF. Sin embargo, no todos estos datos deben ser renderizados de manera simultánea, ya que es necesario aplicar diversos filtros, realizar la paginación, entre otros procesos. Por lo tanto, es crucial identificar qué filas deben ser renderizadas en cada momento.

El primer paso en este proceso consiste en aplicar los filtros especificados por el usuario. Para ello, se crea una variable a la que se asigna el resultado de ejecutar la función `useMemo`. La función `useMemo` recibe dos parámetros: una función y un array de dependencias. Su uso es similar al de `useState`, con la diferencia de que `useMemo` solo ejecutará la función proporcionada como argumento si alguno de los valores del array de dependencias cambia, evitando así cálculos innecesarios que podrían ralentizar el re-renderizado de la página.

En este caso, la función pasada como argumento a `useMemo` realiza las siguientes acciones: copia el valor de *data* en un array auxiliar, filtra este array en función de los campos que el usuario haya especificado (como nombre, *assembly*, fecha de subida o fecha del último informe), permitiendo también la filtración por múltiples campos simultáneamente, y finalmente devuelve el conjunto de filas que cumplen con todos los filtros aplicados.

El array de dependencias de `useMemo` incluye tanto el array *data* como las variables asociadas a los filtros. De esta manera, si los datos de los ficheros cambian o si el usuario modifica los valores de los filtros, la función se ejecutará de nuevo, asegurando que los datos renderizados estén siempre actualizados según los criterios especificados.

Los datos se presentan siempre ordenados según algún campo específico. Una vez que se han aplicado los filtros y se han almacenado los datos filtrados, se crea una nueva

variable destinada a almacenar estos datos ya ordenados. Para calcular este conjunto ordenado, se utiliza nuevamente la función `useMemo`. Esta función recibe como parámetros una función que devuelve los datos ordenados en función del campo seleccionado y un array de dependencias que incluye tanto el campo de ordenación como los datos filtrados. De este modo, los datos ordenados se recalcularán automáticamente si se produce un cambio en los datos filtrados o en el campo de ordenación.

Es importante destacar que se ha optado por realizar primero la operación de filtrado y luego la de ordenación. Esta decisión se basa en el hecho de que el algoritmo de ordenación es más costoso en términos computacionales que el de filtrado. Al aplicar primero el filtrado, se reduce el tamaño del conjunto de datos que debe ser ordenado, lo que disminuye significativamente el tiempo de cálculo necesario para obtener los datos ordenados. En la Figura **??** se muestra la implementación de esta funcionalidad.

Además, la tabla en la que se presentan los datos incluye un sistema de paginación, lo que implica que los datos están divididos en varias páginas y no se muestran todos simultáneamente. Por tanto, es necesario calcular qué datos deben ser mostrados en función de la página actual seleccionada. Para este propósito, también se emplea `useMemo`, utilizando una función que extrae el subconjunto de datos correspondiente a la página actual. El array de dependencias en este caso incluye la página actual y los datos ordenados, garantizando que se muestren los datos correctos en cada página (ver código 3.18).

```
// It sorts the rows of the table
  const sortedItems = useMemo(
    () => sort(filteredItems, sortDescriptor),
    [sortDescriptor, filteredItems],
  );
  // The rows shown on the current page
  const items = useMemo(() => {
    const start = (page - 1) * ROWS_PER_PAGE;
    const end = start + ROWS_PER_PAGE;

    return sortedItems.slice(start, end);
  }, [page, sortedItems]);
```

Listing 3.18: Uso de useState para calcular los datos a mostrar

Como se puede observar, los datos filtrados forman parte del array de dependencias utilizado para calcular los datos ordenados. A su vez, estos datos ordenados forman parte del array de dependencias correspondiente a los datos finales que se mostrarán al usuario. Por lo tanto, si se produce algún cambio en los datos filtrados, se desencadenará una ejecución en cadena que llevará al recalculo de los datos ordenados y, posteriormente, de los datos finales a mostrar.

Una vez obtenidos los datos que deben ser presentados, el siguiente paso consiste en renderizarlos en la interfaz de usuario. La función encargada de renderizar cada celda de la tabla se presenta en el código 3.19.

```
  const renderCell = useCallback((file, columnKey) => {
    const RendererMapping = {
      assembly: <AssemblyCellRenderer file={file} assembly={
         file.assembly} />,
      summary_report_actions: (
        <ReportActionsCellRenderer type={"summary"} file={file
           } />
      ),
```

```
      detailed_report_actions: (
        <ReportActionsCellRenderer type={"detailed"} file={
           file} />
      ),
      actions: <FileActionsCellRenderer file={file} />,
    };

    return RendererMapping[columnKey]
      ? RendererMapping[columnKey]
      : file[columnKey];
  }, []);
```

Listing 3.19: Función que renderiza cada celda

Esta función recibe como parámetros dos elementos: la fila y la columna de la tabla que se va a renderizar. A partir de estos parámetros, se crea un objeto JSON en el que las claves representan los nombres de algunas columnas a renderizar, mientras que los valores son componentes personalizados de *React*. Si el valor de `columnKey` coincide con alguna de las claves del objeto, la función devolverá el componente de *React* correspondiente, que será renderizado posteriormente. En caso de que el valor de `columnKey` no coincida con ninguna clave, esto indica que dicha columna no tiene ningún aspecto personalizado, y la función simplemente devolverá el valor de esa columna, que será renderizado como un texto simple.

Para el renderizado de las otras columnas, se han creado componentes personalizados de *React*, los cuales se describirán a continuación. El primero de estos componentes es `AssemblyCellRender`, encargado de renderizar el valor del campo *assembly*. Este componente es el más sencillo de todos, ya que muestra el valor del *assembly* en formato de texto, similar a las primeras columnas, pero con la diferencia de que aplica un estilo adicional al texto y le asigna un color. El color varía en función del valor del *assembly*, proporcionando así una representación visual distintiva para diferentes valores.

El componente *ReportActionsCellRender* es responsable de renderizar las celdas que contienen las acciones de visualización y descarga de informes, utilizando para ello el microservicio descrito en la Sección 3.4. El contenido de esta sección de la página incluye dos *Tooltips*, cada uno de los cuales contiene un botón, así como un componente modal que se encarga de mostrar el contenido del informe. Es importante destacar que tanto los *Tooltips* como los botones estarán deshabilitados si el *status* del fichero es distinto de `.ºk"`, es decir, si el informe no ha sido generado previamente.

Para implementar las funcionalidades de visualización y descarga de informes, se han definido distintas funciones. En particular, se ha creado una función para la descarga y otra para la visualización, las cuales se invocan al hacer clic en los respectivos botones. Además, se ha implementado una función auxiliar que es invocada por ambas. Esta función auxiliar se encarga de realizar la solicitud al *downloader*, obtener el contenido del fichero y crear una URL temporal que hace referencia a dicho contenido, permitiendo que este pueda ser accedido posteriormente.

```
return fetch("api/report", {
      method: "GET",
      headers: headers,
    })
      .then((response) => {
        if (!response.ok) {
          throw new Error("Network response was not ok");
```

```
        }
        return response.blob();
      })
      .then((blob) => {
        const url = window.URL.createObjectURL(blob);
        setReportPath(url);
        enqueueSnackbar(
          `${type} report from ${file.file_name} can be viewed
             .`,
          {
            variant: "success",
          },
        );
      })
```

Listing 3.20: Fragmento de la función que obtiene el informe.

Como se puede observar en el código 3.20, la función envía la solicitud al microservicio correspondiente, convierte los datos recibidos en un *blob* (Binary Large Object), y luego almacena en la variable *reportPath* una URL temporal y local que hace referencia a dicho *blob*.

Para visualizar el informe, se verifica si la URL ya ha sido generada; en caso contrario, se obtiene la URL mediante la función auxiliar. Una vez que la URL está disponible, se invoca la función `onOpen` para proceder con la visualización del informe, tal como se ilustra en el código 3.21.

```
const viewReport = () => {
  if (!reportPath) {
    getReportPath().then(onOpen);
  } else {
    onOpen();
  }
};
```

Listing 3.21: Función para visualizar el informe.

La función `onOpen` se obtiene al ejecutar la función `useDisclosure` de *React*. `useDisclosure` es un *hook* de *React* que se encarga de gestionar la apertura y cierre de modales. La función `onOpen`, cuando es invocada, tiene la responsabilidad de abrir el modal correspondiente.

El modal, una vez abierto, contiene un elemento `iframe` cuyo atributo `src` apunta a la URL generada que hace referencia al *blob* (ver código 3.22). Esto permite que el contenido del archivo sea visualizado directamente dentro del modal.

```
<ModalBody className={"align-middle grow"}>
    <iframe src={filePath} height={"100%"} />
</ModalBody>
```

Listing 3.22: Contenido del body del modal.

Por último, para renderizar la columna *Actions*, se utiliza el componente `FileActionsCellRender`, el cual contiene tres subcomponentes distintos. Sin embargo, únicamente se renderiza uno de estos subcomponentes dependiendo del *status* del fichero.

Si el *status* del fichero es `executing`, `queue`, o `compressing`, es decir, si el informe se está generando o si el fichero VCF aún no ha terminado de subirse, se renderiza el com-

ponente `ExecutingFileAction`. Este subcomponente se encarga de mostrar simplemente un *label* con un mensaje informativo.

En el caso de que el *status* sea `waiting`, esto indica que el fichero no tiene un *assembly* asignado y que este debe ser introducido manualmente. En este escenario, se renderiza el componente `WaitingFileAction` (ver código 3.23), el cual contiene un botón que, al ser presionado, abre un modal. Este modal incluye un `button group` con dos botones, cada uno correspondiente a un *assembly* diferente (GRCh37 o GRCh38). Al seleccionar uno de estos botones, se asigna el *assembly* al fichero correspondiente.

```
<Button variant={"shadow"} color={"danger"} onPress={onOpen}>
    Set assembly
</Button>
<AssemblyModal
    isOpen={isOpen}
    onOpenChange={onOpenChange}
    onSetAssembly={onSetAssembly}
/>
```

Listing 3.23: Contenido del componente WaitingFileAction.

En el caso de que el *status* del fichero sea `OK` (indica que el informe ha sido generado), `error` (indica que ha ocurrido un error durante la generación del informe), o `no_report` (indica que el informe no ha sido generado), se renderizará el componente por defecto `DefaultFileAction`. Este componente se encarga de mostrar dos botones: uno para generar o regenerar el informe, y otro para eliminar el archivo.

Al presionar el primer botón, se invoca la función `OnClickGenerateReport` (ver código 3.24), que hace uso del microservicio descrito en la Sección 2.4. Esta función envía una solicitud a una ruta local, la cual se encarga de enviar un mensaje al orquestador mediante Apache Kafka. Si la respuesta es positiva, se utiliza la función `mutate` para volver a obtener los datos actualizados. La función `mutate` pertenece a la librería `SWR` y permite solicitar los datos actualizados de inmediato, sin tener que esperar a los intervalos de actualización programados por `useSWR`. Una vez que el informe ha sido generado, el *status* en la base de datos cambiará a `OK`, por lo que será necesario obtener nuevamente los datos para actualizar la interfaz, permitiendo así la renderización de los botones de visualización y descarga.

```
  const onClickGenerateReport = () => {
    const headers = {
      file_name: file.file_name,
    };

    enqueueSnackbar(`Generating reports from ${file.file_name
       }...`, {
      variant: "info",
    });

    fetch("api/vcf/process_vcfs", {
      method: "PUT",
      headers: headers,
    })
      .then((response) => {
        if (!response.ok) {
          throw new Error(response.message);
```

```
        }
        enqueueSnackbar(`${file.file_name} reports are being
            generated`, {
          variant: "info",
        });
      })
      .then(() => {
        return setTimeout(() => {
          return mutate("/api/vcf/get_all_vcfs");
        }, 1500);
      })
      .catch(() => {
        enqueueSnackbar(`Could not generate ${file.file_name}
            reports.`, {
          variant: "error",
        });
      });
  };
```

Listing 3.24: Función que envía la petición de generación.

La página no solo presenta la tabla con los datos y las acciones correspondientes, sino que también incorpora funcionalidades adicionales como el filtrado y la paginación. En la parte superior de la página, se encuentran los filtros, que permiten al usuario realizar búsquedas basadas en cuatro campos distintos: nombre, fecha de subida, fecha del último informe y *assembly*.

Para el filtrado por nombre, se incluye un componente de tipo `Input` que permite al usuario ingresar texto. Este componente posee una propiedad denominada `OnValueChange`, la cual detecta cualquier cambio en el texto ingresado. A esta propiedad se le asigna una función encargada de modificar el valor de la variable que almacena el texto ingresado. Esta acción desencadena la ejecución de la función `useMemo` asociada a los valores filtrados, lo que provoca un nuevo cálculo de los datos filtrados.

La implementación del filtrado por fecha de subida y fecha del último informe sigue un esquema prácticamente idéntico. Ambos campos cuentan con un botón que, al ser presionado, abre un modal compartido. Este modal incluye el componente `Flatpickr` de *React*, que permite seleccionar fechas (ver código 3.25). El componente `Flatpickr` tiene una propiedad `mode` que, al asignársele el valor `range`, permite la selección de un rango de fechas. Una vez seleccionado un rango de fechas, se actualiza el valor de la variable de filtrado correspondiente, lo que a su vez desencadena una actualización de los datos filtrados.

```
<ModalBody>
    <Flatpickr
        value={dateRange}
        options={{
            dateFormat: "d-m-Y",
            mode: "range",
            inline: true,
        }}
        onChange={(range) => {
            if (range.length === 2) {
                setDateRange(range);
                onClose();
```

```
                }
            }}
        />
</ModalBody>
```

Listing 3.25: Cuerpo del modal selector de fechas.

Para el filtrado por *assembly*, se emplea el elemento `Dropdown`, diseñado para la creación de menús desplegables. Este componente incluye un botón dentro de un elemento `DropdownTrigger`, el cual, al ser presionado, activa el elemento `DropdownMenu`.

Dentro del menú desplegable, se itera sobre un *array* de opciones, generando un `item` en el menú para cada una de las opciones disponibles. De esta manera, el usuario puede seleccionar el *assembly* deseado directamente desde el menú desplegable.

React proporciona un componente denominado `Pagination`, que facilita la implementación de la paginación en una aplicación. Este componente se integra en la interfaz junto con botones que permiten la navegación a la página anterior o siguiente.

Al presionar uno de estos botones, se invoca una función específica que se encarga de incrementar o decrementar en uno el valor de la variable `page`. La modificación del valor de esta variable desencadena el recálculo de las filas que deben mostrarse en la tabla.

En el código 3.26 se puede observar la implementación detallada de este componente de paginación.

```
<Pagination {...paginationParameters} />
<div className="hidden sm:flex w-[30%] justify-end gap-2">
    <Button
        isDisabled={pages === 1}
        size="sm"
        variant="flat"
        onPress={onPreviousPage}
    >
        Previous
    </Button>
    <Button
        isDisabled={pages === 1}
        size="sm"
        variant="flat"
        onPress={onNextPage}
    >
        Next
    </Button>
```

Listing 3.26: Implementación de la paginación.

Por último, queda describir la implementación de la pantalla de subida de ficheros. Se usa la función *useDropzone* de react para implementar una zona de arrastre y suelta de ficheros (figura **??**), esta función se encarga de implementar toda la lógica necesaria. Además hay un botón *select files* que al ser pulsado, se invoca a la función *open* obtenida de *useDropzone* que abre el navegador de archivos de forma que se pueda también subir archivos sin arrastrar y soltar. Cada vez que se selecciona un archivo, se muestra información sobre él como el nombre, el tipo o el tamaño, además se le añade un botón que permita eliminar el archivo en el caso en el que se haya seleccionado por error.

Por último, se incluye un botón *Upload* que al ser presionado, invoca una función que recoge los archivos seleccionados y haciendo uso del microservicio del apartado 3.3 sube los archivos al servidor.

Finalmente, es necesario describir la implementación de la pantalla de subida de ficheros. Para este propósito, se utiliza la función `useDropzone` de *React*, la cual permite implementar de manera eficiente una zona de arrastre y suelta de ficheros (ver código 3.27). Esta función se encarga de gestionar toda la lógica necesaria para el manejo de los archivos que se arrastran y sueltan en dicha zona.

Además, se ha añadido un botón etiquetado como `Select Files`. Al ser pulsado, se invoca la función `open`, obtenida a través de `useDropzone`, la cual abre el explorador de archivos del sistema, permitiendo al usuario seleccionar archivos sin necesidad de arrastrarlos y soltarlos. Cada vez que se selecciona un archivo, se muestra información relevante, como el nombre del archivo, el tipo de archivo y su tamaño. Adicionalmente, se incluye un botón asociado a cada archivo seleccionado que permite eliminarlo en caso de que haya sido añadido por error.

Finalmente, se ha implementado un botón etiquetado como `Upload`, que, al ser presionado, invoca una función específica encargada de recopilar los archivos seleccionados y, mediante el microservicio descrito en la Sección 3.3, proceder con la subida de los archivos al servidor.

```
1  const { isDragActive, getRootProps, getInputProps, open } =
       useDropzone({
2    accept: {
3      "file/vcf": [".vcf"],
4    },
5    noKeyboard: true,
6    onDrop,
7  });
```

Listing 3.27: Uso de la función useDropzone.

3.5.3. Despliegue

Para el despliegue se ha optado por utilizar Vercel, Vercel es una plataforma de despliegue y hosting optimizada para aplicaciones web front-end y full-stack. Fue desarrollada inicialmente como Zeit Now y es especialmente conocida por su integración con el framework de desarrollo Next.js, aunque también soporta otras tecnologías populares como React, Vue.js, Angular y Svelte. Vercel ofrece despliegue instantáneo, integración con repositorios de GitHub, GitLab y Bitbucket, escalabilidad automática, una CDN global para tiempos de carga rápidos, previsualizaciones automáticas de despliegue y soporte para funciones serverless.

Usar Vercel para desplegar aplicaciones web tiene varias ventajas. Primero, su facilidad de uso permite a los desarrolladores centrarse en el código sin preocuparse por la infraestructura. Con su integración continua, cada cambio en el repositorio desencadena un despliegue automático, asegurando que la aplicación esté siempre actualizada. Además, las aplicaciones se benefician de una red de distribución de contenido global, lo que garantiza un rendimiento optimizado y tiempos de carga rápidos en cualquier ubicación. Las previsualizaciones de despliegue facilitan la colaboración entre desarrolladores y diseñadores, permitiendo revisar cambios antes de implementarlos en producción. La escalabilidad automática maneja grandes volúmenes de tráfico sin configuraciones

adicionales, y el soporte para funciones serverless permite ejecutar lógica del lado del servidor sin gestionar servidores.

Para desplegar una aplicación en Vercel, es necesario poseer una cuenta, Vercel permite iniciar sesión con una cuenta de GitLab, GitHub o Bitbucket (figura 3.6), por lo que es bastante recomendable iniciar sesión con la cuenta del repositorio en el que está desarrollada la aplicación, de forma que se pueda importar directamente.

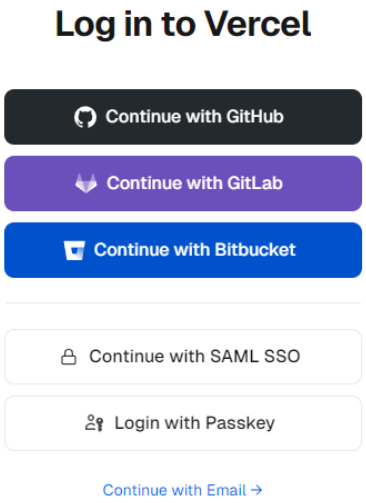

Figura 3.6: Pantalla de inicio de sesión en Vercel.

Para crear un nuevo proyecto, en la pantalla principal se pulsa sobre *Add New... ->Project* (figura 3.7).

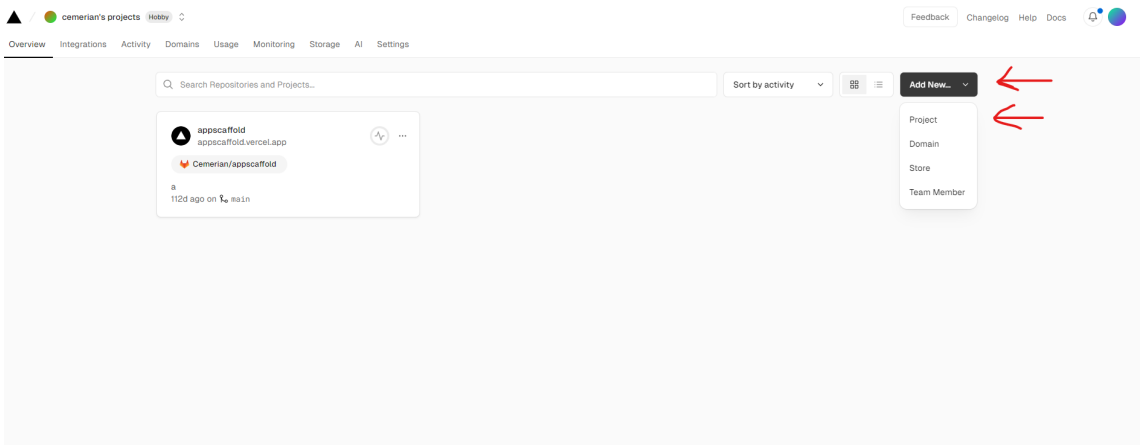

Figura 3.7: Pantalla principal en Vercel.

Si se ha iniciado sesión con la cuenta del repositorio del proyecto, el proyecto debería aparecer en la siguiente pantalla, al pulsar el botón *Import* se importa el proyecto (figura 3.8)

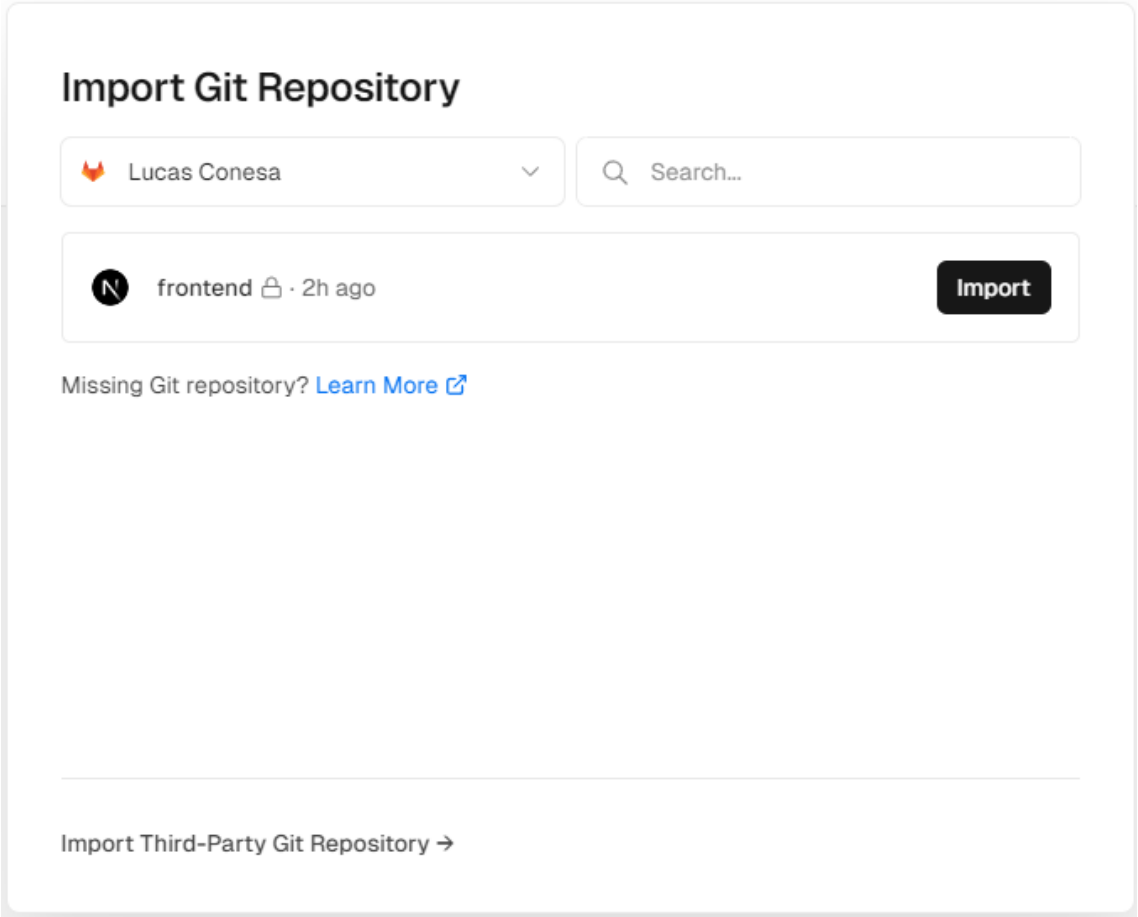

Figura 3.8: Pantalla para importar un proyecto en Vercel.

Por último, se le da nombre al proyecto, se establece Next.js como *framework*(por defecto) y se configuran las variables de entorno (se puede copiar y pegar un archivo .env en caso de que ya exista uno), pulsamos sobre *Deploy* y la aplicación se desplegará correctamente. En la figura 3.9 se puede observar ésta última pantalla de configuración.

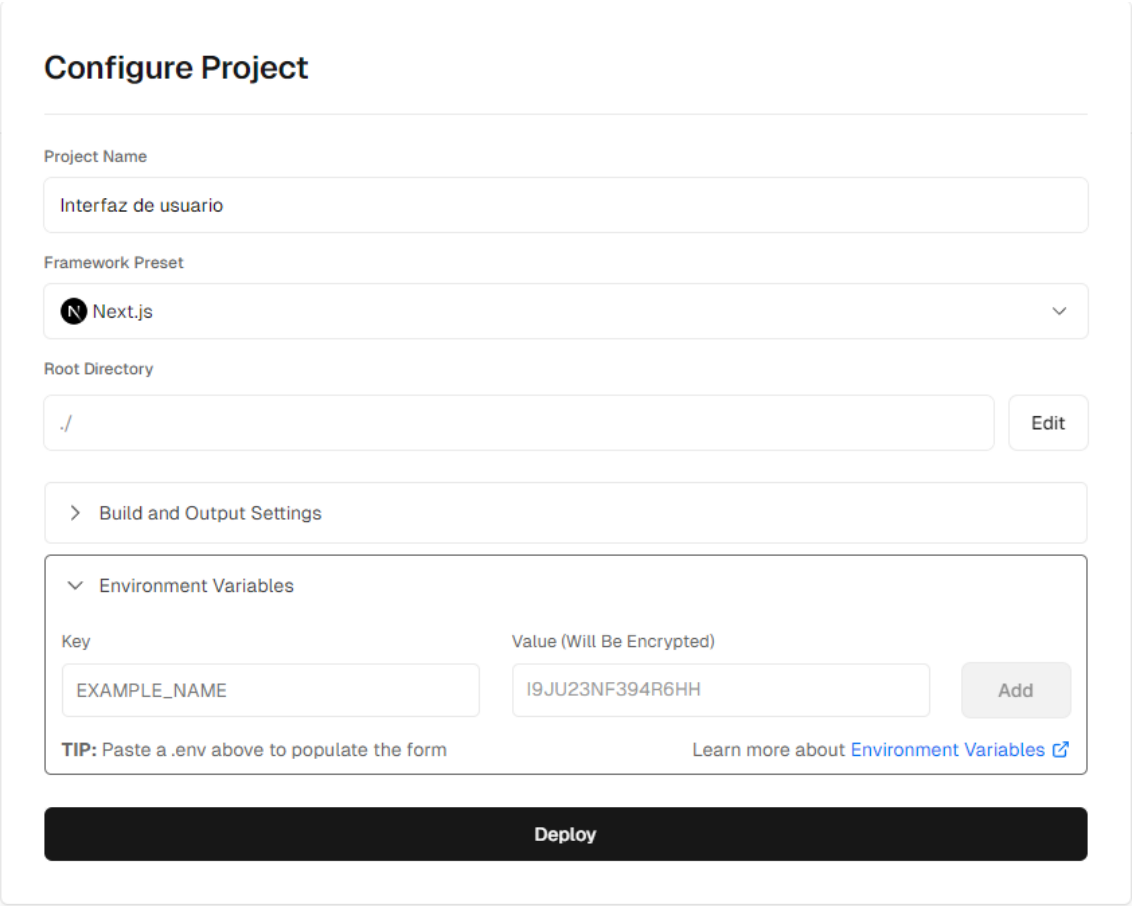

Figura 3.9: Pantalla de configuración del proyecto.

3.6 Respuesta a preguntas de investigación

¿Qué componentes principales debe tener la arquitectura? Los componentes principales del sistema incluyen el *Uploader*, el *Downloader* y el Orquestador, junto con la interfaz de usuario o *frontend*, que permite la interacción directa del usuario con el sistema. Estos elementos son esenciales para la gestión de la subida y descarga de archivos, así como para la coordinación de las tareas internas. Además, el sistema emplea *Apache Kafka* para la comunicación entre componentes, *Workers* para realizar las tareas computacionales más intensivas, *Redis* para la gestión eficiente de colas de trabajo, y una base de datos para almacenar metadatos y otra información relevante.

¿Qué stack tecnológico es el más adecuado? La elección de JavaScript y Node.js como el lenguaje de programación principal se justifica por su alta eficiencia en operaciones de entrada/salida (I/O) y su naturaleza asíncrona, que facilitan la integración con distintos componentes del sistema. Además, permite unificar el desarrollo *frontend* y *backend* bajo un mismo lenguaje, mejorando la coherencia del código y la productividad del equipo. El amplio ecosistema de npm y la activa comunidad de Node.js proporcionan soporte continuo, recursos abundantes y actualizaciones regulares, asegurando que el proyecto se mantenga actualizado y funcional a largo plazo.

¿Cómo realizar el despliegue de la arquitectura? Dado que la arquitectura del sistema se basa en microservicios, la estrategia de despliegue varía según el entorno en el

que se alojan los componentes. Los componentes que residen en la nube ya están desplegados, lo que elimina la necesidad de realizar acciones adicionales de implementación para estos elementos específicos. En cuanto a los microservicios que deben ejecutarse localmente, su despliegue se realiza de manera eficiente y sencilla utilizando *npm*, lo que facilita la configuración y gestión del entorno local. Por otro lado, la interfaz de usuario, una parte crucial de la interacción del usuario con el sistema, se desplegará en la nube a través de Vercel, asegurando así un acceso rápido, seguro y escalable desde cualquier ubicación.

CAPÍTULO 4
Validación de la solución

La validación de una aplicación web es un proceso crucial en el ciclo de desarrollo de software, ya que garantiza que la herramienta no solo funcione correctamente, sino que también cumpla con las expectativas y necesidades de los usuarios. Para llevar a cabo esta validación, hemos empleado una metodología basada en la observación directa y la evaluación mediante encuestas de satisfacción basadas en la escala Likert. Este capítulo detalla el proceso de validación que hemos realizado, incluyendo la metodología utilizada, la justificación de la elección de la escala Likert, y los resultados preliminares obtenidos.

4.1 Metodología

Para validar la efectividad y aceptación de la aplicación web, seleccionamos un grupo de 10 usuarios representativos del público objetivo. Estos usuarios fueron escogidos cuidadosamente para asegurar una variedad en términos de ocupaciones, así como de niveles de experiencia académica y profesional, con el fin de obtener una muestra diversificada que reflejara las diferentes perspectivas y necesidades del público general.

El proceso de validación se dividió en dos fases distintas. La primera fase consistió en la observación directa de los usuarios mientras interactuaban con la aplicación, utilizando el método de *pensar en voz alta* (*think-aloud*). En esta técnica, a cada usuario se le pidió que verbalizara sus pensamientos, dudas, y decisiones mientras navegaba por la aplicación. Este método es especialmente útil para identificar problemas de usabilidad, ya que proporciona una visión directa de los procesos cognitivos del usuario durante su interacción con el sistema. Los comentarios espontáneos y las reflexiones de los usuarios permiten detectar obstáculos en la navegación, confusiones en la interfaz, o cualquier otro aspecto que pueda afectar la experiencia de uso.

La segunda fase del proceso de validación consistió en la aplicación de una encuesta estructurada basada en la escala Likert para obtener una evaluación cuantitativa de la percepción de los usuarios. La encuesta Likert es una herramienta ampliamente utilizada en investigaciones sociales y de mercado, que permite medir actitudes o percepciones en función de un conjunto de afirmaciones relacionadas con la aplicación. Cada afirmación se califica en una escala de cinco puntos, que varía desde "totalmente de acuerdo"hasta "totalmente en desacuerdo". Este enfoque cuantitativo proporciona una medida objetiva de la satisfacción del usuario, permitiendo identificar tendencias generales en la percepción del grupo de usuarios evaluados.

4.1.1. Perfil de los Usuarios

El grupo de usuarios estuvo compuesto por los perfiles que se muestran en la tabla 4.1:

Usuario	Perfil
U1	Doctor en Informática
U2	Doctor en Informática
U3	Doctor en Informática
U4	Doctor en Informática
U5	Estudiante de doctorado en el programa de tecnologías para la salud y el bienestar, con un grado en Ingeniería Biomédica.
U6	Estudiante de doctorado en el programa de tecnologías para la salud y el bienestar, con un grado en Física.
U7	Técnico superior con grado en Ingeniería Informática.
U8	Técnico superior con grado en Ingeniería Informática.
U9	Técnico superior con grado en Ingeniería Informática.
U10	Estudiante de máster en Ingeniería Biomédica, con un grado en Ingeniería Biomédica.

Tabla 4.1: Perfiles de los usuarios.

Este grupo fue seleccionado para proporcionar una amplia perspectiva sobre la aplicación, cubriendo tanto a usuarios con un profundo conocimiento técnico como a aquellos con formación interdisciplinaria en áreas relacionadas con la salud y la tecnología.

4.1.2. Fase de Observación Directa

En la primera fase, cada usuario interactuó con la aplicación en un entorno controlado y realizó un conjunto de tareas predeterminadas que simulan escenarios de uso real. Estas tareas fueron diseñadas para cubrir las principales funcionalidades de la aplicación y evaluar su usabilidad y eficiencia. Las tareas que los usuarios realizaron fueron las siguientes:

- Iniciar sesión en la aplicación.
- Subir 4 ficheros VCF proporcionados (VCF1, VCF2, VCF3, VCF4).
- Generar 1 informe basado en el fichero VCF1.
- Visualizar el informe corto generado a partir de VCF1.
- Descargar el informe largo generado a partir de VCF1.
- Regenerar 4 informes siguiendo el orden (VCF1, VCF2, VCF3, VCF4).
- Descargar el informe corto del fichero VCF4.

Durante la ejecución de estas tareas, se les pidió a los usuarios que verbalizaran sus pensamientos, impresiones y dificultades mientras usaban la herramienta. Este enfoque, conocido como *think-aloud*, permite a los observadores captar de manera inmediata y cualitativa las experiencias y posibles obstáculos que enfrentan los usuarios. Los observadores documentaron tanto las verbalizaciones de los usuarios como sus acciones y reacciones, identificando así problemas de usabilidad, áreas de mejora y aspectos positivos de la interfaz.

4.1.3. Fase de Evaluación Cuantitativa

Tras completar las tareas, se solicitó a los usuarios que respondieran a una encuesta estructurada basada en la escala Likert de 5 puntos, diseñada para evaluar tres dimensiones clave de la experiencia del usuario con la aplicación:

- **Perceived Ease of Use (PEOU)**: Esta categoría mide la facilidad percibida de uso de la aplicación. Se incluyeron las siguientes afirmaciones:
  - Considero que subir un fichero es una tarea sencilla.
  - Considero fácil encontrar los ficheros que busco.
  - Es fácil recordar cómo se realiza cada acción/tarea.
  - Descargar/visualizar informes PDF me resulta sencillo.
  - Generar un informe me resulta sencillo.
  - En general, considero que la aplicación es intuitiva.
- **Perceived Usefulness (PU)**: Esta categoría evalúa la utilidad percibida de la aplicación, es decir, en qué medida la aplicación mejora el rendimiento y facilita las tareas del usuario. Las afirmaciones incluidas fueron:
  - Los ficheros VCF se suben rápidamente.
  - Los informes en PDF se generan en menos tiempo que usando otros métodos.
  - Los informes muestran información relevante.
  - La información se encuentra estructurada y organizada.
  - La aplicación me permite ser más productivo en mi trabajo/mejora mi rendimiento general.
- **Intention to Use (ITU)**: Esta categoría refleja la intención del usuario de utilizar la aplicación en el futuro y su disposición a recomendarla a otros. Las afirmaciones para esta categoría fueron:
  - Usaré esta aplicación cuando esté disponible.
  - Recomendaría a mi jefe de proyecto el uso de esta aplicación.

Cada afirmación se evaluó en una escala de 1 a 5, donde 1 representa “Muy en desacuerdo” y 5 representa “Muy de acuerdo”.

4.2 Análisis de Resultados

Después de que los usuarios completaran las tareas y respondieran la encuesta basada en la escala Likert, se procedió a analizar los resultados obtenidos en las tres dimensiones clave: *Perceived Ease of Use (PEOU)*, *Perceived Usefulness (PU)* e *Intention to Use (ITU)*. A continuación, se presentan los resultados y un análisis detallado de los mismos.

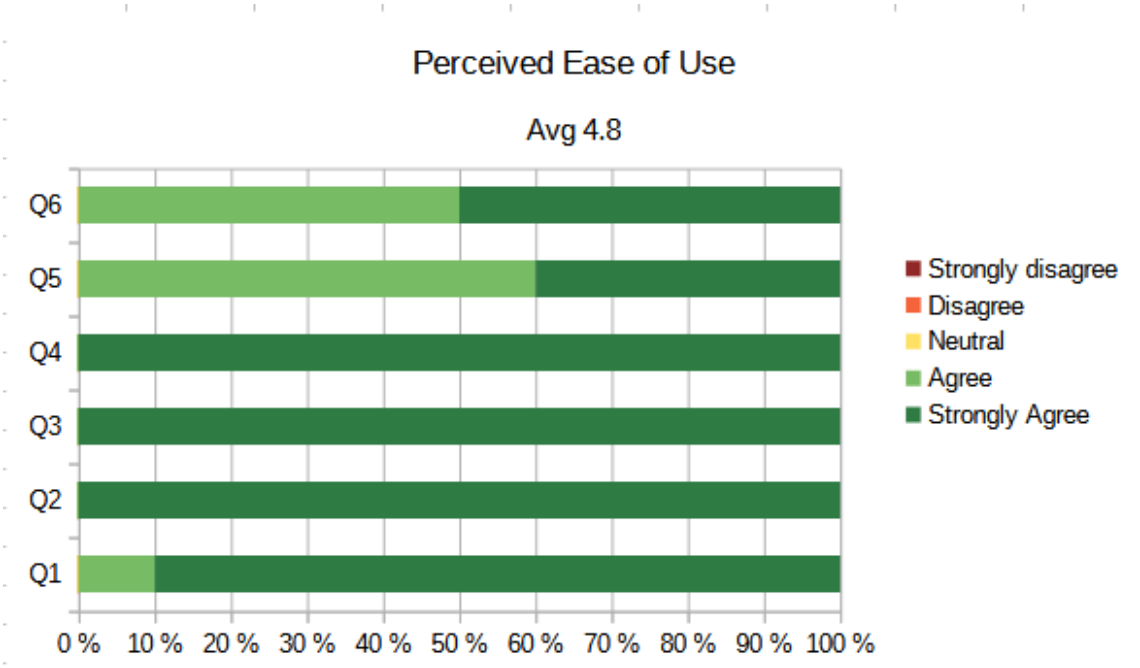

Figura 4.1: Gráfico de resultados PEOU.

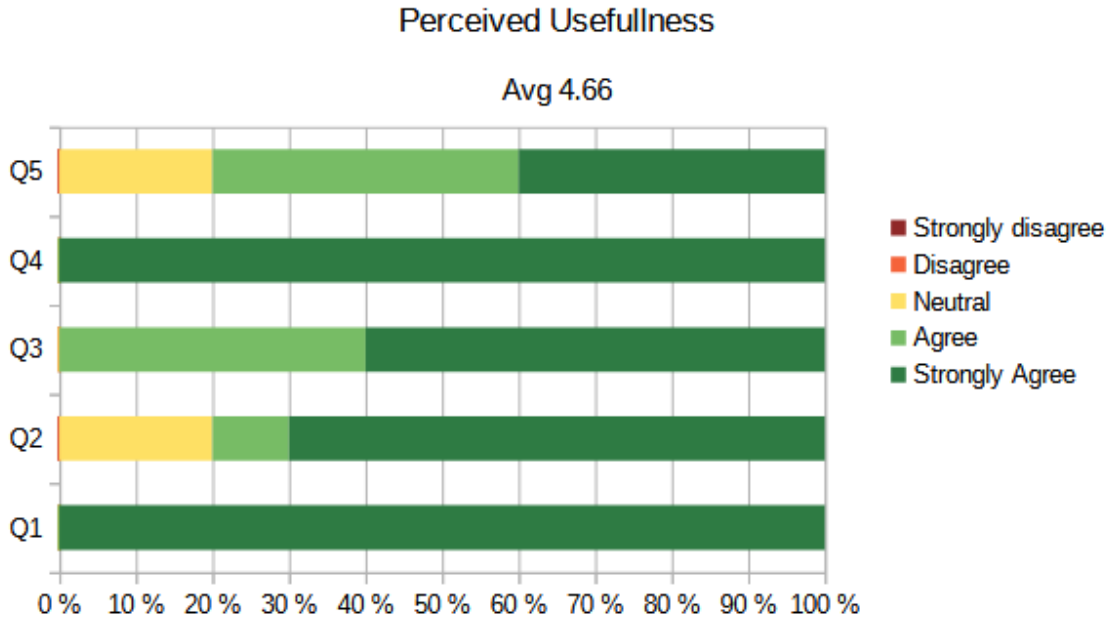

Figura 4.2: Gráfico de resultados PU.

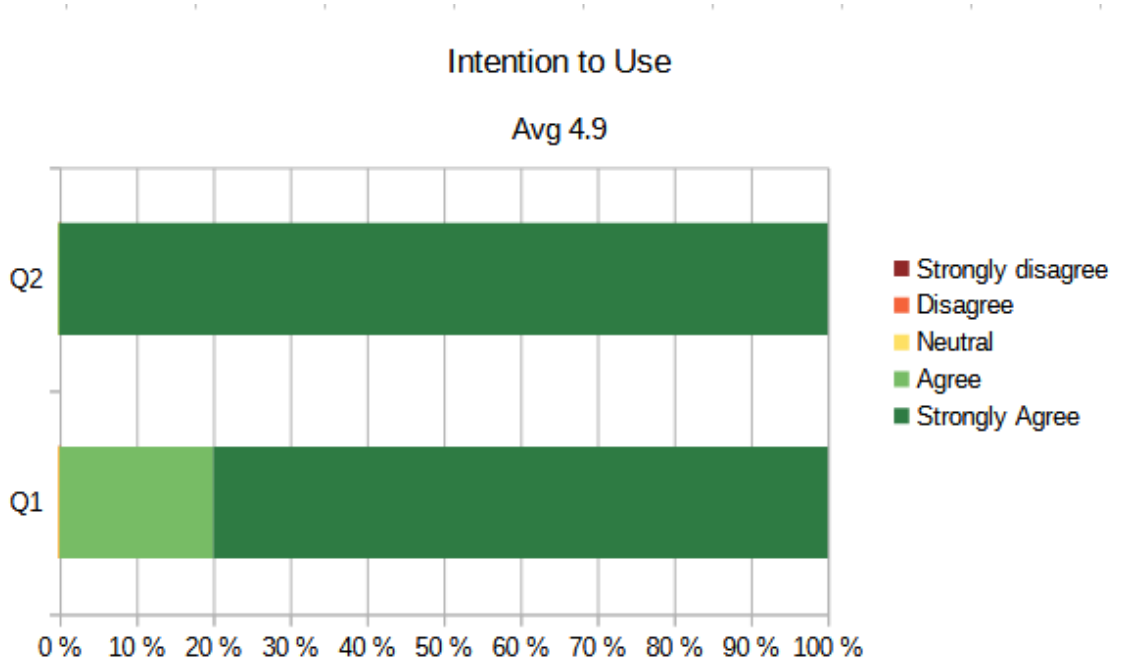

Figura 4.3: Gráfico de resultados ITU

Al analizar los resultados en función del perfil de los usuarios, se observan algunas tendencias interesantes que permiten una interpretación más profunda de los datos:

- **PEOU (Facilidad Percibida de Uso)(Figura 4.1)**: La media de 4.8 refleja una percepción generalmente alta de la facilidad de uso. Sin embargo, al desglosar los resultados, se puede observar que los técnicos superiores (U7, U8, U9) otorgaron puntuaciones ligeramente menores en aspectos como la facilidad para recordar cómo realizar tareas y la facilidad para encontrar lo que buscan. Específicamente, en la afirmación "Es fácil recordar cómo se realiza cada acción/tarea", los técnicos otorgaron un promedio de 4, mientras que los doctores en informática y los estudiantes de doctorado otorgaron un 5. Esta diferencia puede deberse a que los técnicos están menos familiarizados con las tareas específicas que se realizan en la herramienta y, por lo tanto, encuentran más difícil recordar los pasos a seguir. De manera similar, los técnicos consideraron que la aplicación no es tan intuitiva como lo percibieron los doctores, probablemente debido a su menor familiaridad con el dominio específico de la aplicación.

- **PU (Utilidad Percibida)(Figura 4.2)**: Con una media de 4.66, la utilidad percibida de la aplicación es alta en todos los grupos. Sin embargo, se observa que los doctores en informática valoraron ligeramente mejor la utilidad de la aplicación que los técnicos superiores. Esta tendencia puede estar relacionada con la mayor experiencia y conocimiento de los doctores, quienes tienen una visión más amplia del valor que la aplicación puede aportar a su trabajo. Esta mayor comprensión del potencial de la herramienta podría influir en su valoración más positiva.

- **ITU (Intención de Uso)(Figura 4.3)**: La media de 4.9 en esta categoría muestra que prácticamente todos los usuarios tienen una alta intención de seguir utilizando la aplicación en el futuro y recomendarla a otros. Esta alta intención de uso es un indicador positivo de la aceptación de la herramienta entre los usuarios, independientemente de sus niveles de experiencia.

En general, los resultados obtenidos son muy positivos, con medias cercanas al 5 en todas las categorías evaluadas. La aplicación es percibida como intuitiva, útil y fácil de usar por la mayoría de los usuarios, lo que sugiere que cumple con los requisitos esperados y que su diseño está alineado con las necesidades y expectativas de los usuarios.

No obstante, las observaciones realizadas indican que existe una diferencia en la percepción de la aplicación entre usuarios con diferentes niveles de experiencia. Los usuarios con mayor experiencia (doctores en informática y estudiantes de doctorado) parecen tener una visión más favorable de la aplicación, lo cual puede estar influido por su comprensión más profunda del dominio y de las funcionalidades que la herramienta ofrece. Por otro lado, los técnicos superiores, aunque también perciben la aplicación de manera positiva, encuentran más difícil recordar cómo se realizan ciertas tareas y no la consideran tan intuitiva como sus colegas más experimentados. Esto sugiere que podría ser beneficioso proporcionar guías de uso o tutoriales adicionales que ayuden a los usuarios menos familiarizados con el dominio a utilizar la aplicación de manera más efectiva.

En conclusión, aunque se han identificado áreas de mejora, los resultados indican que la aplicación tiene un alto potencial de aceptación y que satisface las necesidades de los usuarios objetivo.

4.3 Respuesta a preguntas de investigación

¿En qué medida la herramienta resulta útil para los usuarios finales? Después de presentar la versión final de la aplicación a un grupo seleccionado de usuarios y solicitarles que completaran una encuesta de tipo Likert para evaluar distintos aspectos, como la utilidad percibida, la facilidad de uso y la intención de uso futuro, se han obtenido resultados notablemente positivos. Estos resultados sugieren de manera clara que la herramienta desarrollada es percibida como una solución eficaz y valiosa por parte de los usuarios, lo que confirma su relevancia y potencial en el contexto para el cual fue diseñada.

CAPÍTULO 5
Conclusiones

En el marco del presente Trabajo Fin de Grado, cuyo objetivo principal fue diseñar y desarrollar una plataforma orientada a médicos y genetistas para la realización de análisis de variaciones genómicas, se han alcanzado hitos importantes que reflejan el éxito del proceso de desarrollo. Desde el inicio, se abordó la comprensión del dominio de los usuarios, identificando las tareas clave que realizan y los desafíos que enfrentan. En este contexto, se determinó que el formato VCF es el estándar óptimo para el manejo de datos genómicos, dado su amplio uso en la investigación genética y su capacidad para representar variantes genómicas de manera eficiente y estandarizada. Además, se identificó la necesidad crítica de garantizar la seguridad y privacidad de los datos, implementando medidas robustas para proteger información altamente sensible.

La elección de una arquitectura de microservicios ha sido un acierto, permitiendo una modularidad que facilita tanto el desarrollo como la escalabilidad del sistema. Componentes esenciales como el *Uploader*, el *Downloader*, y el Orquestador se integraron eficientemente, mientras que tecnologías como *Apache Kafka* y *Redis* aseguraron una comunicación fluida entre los microservicios y una gestión eficaz de las tareas. Node.js, junto con su ecosistema de *npm*, proporcionó una base sólida que unifica el desarrollo *frontend* y *backend*, mejorando la coherencia del código y la productividad del equipo.

En cuanto a la implementación y despliegue, los componentes alojados en la nube y aquellos desplegados localmente se integraron con éxito, utilizando herramientas como *npm* para los microservicios locales y Vercel para la interfaz de usuario. Esto garantizó un entorno de operación robusto y accesible, capaz de responder a las necesidades de los usuarios finales.

Los resultados obtenidos tras la validación con un grupo de usuarios finales han sido altamente satisfactorios. Las encuestas tipo Likert revelaron una percepción positiva en términos de utilidad, facilidad de uso e intención de uso futuro, lo que confirma que la plataforma cumple con las expectativas y necesidades del público objetivo. Estos resultados no solo validan el enfoque adoptado durante el desarrollo, sino que también destacan el potencial de la plataforma para ser adoptada a mayor escala en el ámbito médico y genético.

De cara al desarrollo futuro, se identifican varias áreas de mejora y expansión. La incorporación de nuevas funcionalidades, la optimización de los microservicios existentes, y la ampliación de las capacidades de análisis genómico son algunas de las direcciones en las que se podría avanzar. Asimismo, la posibilidad de integrar la plataforma con otros sistemas y bases de datos genómicas, así como la mejora continua de las medidas de seguridad, serán cruciales para mantener y aumentar la relevancia de la plataforma en un entorno tecnológico y científico en constante evolución.

En conclusión, el proyecto ha logrado cumplir con los objetivos planteados, desarrollando una plataforma robusta y funcional que responde a las necesidades específicas de médicos y genetistas. Los resultados positivos obtenidos durante la fase de validación indican que la herramienta tiene un alto potencial de impacto en el campo del análisis genómico. Con vistas al futuro, se presenta una oportunidad significativa para seguir mejorando y expandiendo la plataforma, asegurando que continúe siendo una herramienta de valor en la investigación genética y la medicina personalizada.

Bibliografía

[1] Mark G Kris et al. "Using multiplexed assays of oncogenic drivers in lung cancers to select targeted drugs". En: *Nature Medicine* 22.11 (2016), págs. 1354-1360.

[2] Bruce Alberts et al. *Molecular Biology of the Cell*. 4th. Chapter 4, DNA: The Genetic Material. New York: Garland Science, 2002. URL: https://www.ncbi.nlm.nih.gov/books/NBK26870/.

[3] JD Watson y FH Crick. "Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid". En: *Nature* 171.4356 (1953), págs. 737-738.

[4] Eric S Lander et al. "Initial sequencing and analysis of the human genome". En: *Nature* 409.6822 (2001), págs. 860-921.

[5] Peter H Sudmant et al. "Diversity of human copy number variation and multicopy genes". En: *Science* 330.6004 (2010), págs. 641-646.

[6] Peter M Visscher, William G Hill y Naomi R Wray. "Heritability in the genomics era–concepts and misconceptions". En: *Nat Rev Genet* 9.4 (2008), págs. 255-266.

[7] Teri A Manolio et al. "Finding the missing heritability of complex diseases". En: *Nature* 461.7265 (2009), págs. 747-753.

[8] Amit V Khera et al. "Genome-wide polygenic scores for common diseases identify individuals with risk equivalent to monogenic mutations". En: *Nature genetics* 50.9 (2018), págs. 1219-1224.

[9] Anubha Mahajan et al. "Fine-mapping type 2 diabetes loci to single-variant resolution using high-density imputation and islet-specific epigenome maps". En: *Nature genetics* 50.11 (2018), págs. 1505-1513.

[10] Georgina V Long et al. "Dabrafenib and trametinib versus dabrafenib and placebo for Val600 BRAF-mutant melanoma: a multicentre, double-blind, phase 3 randomised controlled trial". En: *The Lancet* 391.10123 (2018), págs. 924-935.

[11] Lucia A Hindorff et al. "Potential etiologic and functional implications of genome-wide association loci for human diseases and traits". En: *Proc Natl Acad Sci U S A* 106.23 (2009), págs. 9362-9367.

[12] Daniel G MacArthur et al. "Guidelines for investigating causality of sequence variants in human disease". En: *Nature* 508.7497 (2014), págs. 469-476.

[13] William McLaren et al. "The ensembl variant effect predictor". En: *Genome biology* 17.1 (2016), págs. 1-14.

[14] Melissa J Landrum et al. "ClinVar: public archive of relationships among sequence variation and human phenotype". En: *Nucleic acids research* 42.D1 (2014), págs. D980-D985.

[15] Sue Richards et al. "Standards and guidelines for the interpretation of sequence variants: a joint consensus recommendation of the American College of Medical Genetics and Genomics and the Association for Molecular Pathology". En: *Genetics in Medicine* 17.5 (2015), págs. 405-424.

[16] Laura M Amendola et al. "Performance of ACMG-AMP variant-interpretation guidelines among nine laboratories in the Clinical Sequencing Exploratory Research Consortium". En: *The American Journal of Human Genetics* 98.6 (2016), págs. 1067-1076.

[17] R. J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Google-Books-ID: xLKLBQAAQBAJ, cit. on pp. 11, 19. Springer, nov. de 2014. ISBN: 978-3-662-43839-8.

[18] "The Variant Call Format (VCF) Version 4.2 Specification". En: (2022).

[19] William Stallings. *Computer Security: Principles and Practice*. 4th. Pearson, 2017.

[20] Bruce Schneier. *Data and Goliath: The Hidden Battles to Collect Your Data and Control Your World*. W.W. Norton & Company, 2015.

[21] Daniel J. Solove. *Understanding Privacy*. Harvard University Press, 2021.

[22] Helen Nissenbaum. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford Law Books, 2010.

[23] Dieter Gollmann. *Computer Security*. 3rd. Wiley, 2011.

[24] Charles P. Pfleeger y Shari Lawrence Pfleeger. *Security in Computing*. 5th. Pearson, 2015.

[25] "A Guide to Functional Requirements (with Examples)". En: *nuclino* (unknown). URL: https://www.nuclino.com/articles/functional-requirements.

APÉNDICE A
Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No Procede
Fin de la pobreza.				X
Hambre cero.				X
Salud y bienestar.	X			
Educación de calidad.	X			
Igualdad de género.		X		
Agua limpia y saneamiento.				X
Energía asequible y no contaminante.		X		
Trabajo decente y crecimiento económico.		X		
Industria, innovación e infraestructuras.	X			
Reducción de las desigualdades.		X		
Ciudades y comunidades sostenibles.		X		
Producción y consumo responsables.		X		
Acción por el clima.				X
Vida submarina.				X
Vida de ecosistemas terrestres.				X
Paz, justicia e instituciones sólidas.	X			
Alianzas para lograr objetivos.	X			

A continuación, se exponen cuáles son los objetivos de desarrollo sostenible, pertenecientes a la agenda 2030 de las Organización de las Naciones Unidas (ONU) que se busca cumplir a lo largo de este trabajo

- **ODS 3: Salud y Bienestar**
  Descripción: Garantizar una vida sana y promover el bienestar para todos en todas las edades.
  Contribución: La aplicación puede mejorar el acceso a datos genómicos y análisis clínicos, facilitando diagnósticos más rápidos y precisos, lo que contribuye a mejorar la atención médica y el bienestar de los pacientes. Además, al centralizar la

información en la nube, se facilita la colaboración entre profesionales de la salud, lo que puede resultar en una mejor calidad de la atención y decisiones médicas más informadas.

- **ODS 4: Educación de Calidad**
  Descripción: Garantizar una educación inclusiva y equitativa de calidad y promover oportunidades de aprendizaje durante toda la vida para todos.
  Contribución: La plataforma podría ser utilizada para fines educativos, como la formación de profesionales de la salud en genética y bioinformática, ofreciendo acceso a datos genéticos reales y herramientas de análisis, lo que contribuiría al desarrollo de habilidades y conocimientos en el sector.

- **ODS 9: Industria, Innovación e Infraestructura**
  Descripción: Construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible, y fomentar la innovación.
  Contribución: La aplicación es un ejemplo de innovación en el campo de la salud, utilizando tecnología moderna (como la nube y el análisis de datos genómicos) para mejorar los procesos clínicos. Esto también contribuye al desarrollo de infraestructura digital en el sector de la salud, promoviendo la transformación digital en la medicina.

- **ODS 16: Paz, Justicia e Instituciones Sólidas**
  Descripción: Promover sociedades pacíficas e inclusivas para el desarrollo sostenible, proporcionar acceso a la justicia para todos y construir instituciones eficaces, responsables e inclusivas a todos los niveles.
  Contribución: Al garantizar la privacidad y seguridad de los datos clínicos y genómicos almacenados en la nube, la aplicación contribuye a la creación de instituciones sólidas y responsables en el manejo de información sensible, lo que es esencial para mantener la confianza de los usuarios y garantizar la justicia y la protección de los datos.

- **ODS 17: Alianzas para lograr los Objetivos**
  Descripción: Fortalecer los medios de implementación y revitalizar la Alianza Mundial para el Desarrollo Sostenible.
  Contribución: La aplicación podría fomentar la colaboración entre diferentes instituciones médicas, académicas y de investigación, facilitando el intercambio de datos y recursos, lo que contribuye a fortalecer alianzas globales en el ámbito de la salud y la investigación genética.