Práctica Profesional Supervisada

Scholz, Veronica Noemi

Aplicación web para el análisis de datos genómicos relacionados con la Vitamina D

Instituto de Ingeniería y Agronomía

Carrera: Ingeniería en Informática

2025

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Ingeniería en Informática

INFORME FINAL DEL PROYECTO INTEGRADOR PROFESIONALIZANTE

Aplicación web para el análisis de datos genómicos relacionados con la Vitamina D

Florencio Varela, diciembre 2025

Estudiante

Veronica Noemi Scholz

vero.scholz@gmail.com

Coordinador de Ingeniería en Informática

Dr. Ing. Martín Morales

martin.morales@unaj.edu.ar

Tutor

Dr. H. Ariel Alvarez

haalvarez@unaj.edu.ar

Tutor

Ing. Maximiliano Olivera

leolivera@unaj.edu.ar

Docente

Dr. Ing. Ramiro Miguel Irastorza

rirastorza@unaj.edu.ar

En colaboración con:

Dr. Eduardo I. Howard

howard@iflysib.unlp.edu.ar

Resumen

Este proyecto presenta una aplicación web, para el análisis de datos genómicos, que nació con la idea de investigar cómo ciertas variaciones genéticas pueden estar relacionadas con condiciones como el raquitismo en diferentes poblaciones. A partir de una secuencia ingresada por el usuario, la plataforma permite compararla con secuencias y estructuras de referencia, identificando posibles factores genéticos que expliquen diferencias fenotípicas.

Mediante una arquitectura de microservicios contenedorizados con Docker, coordinados por un backend en .NET y un frontend en React con TypeScript, los servicios internos, como Bioconductor, Biopython y BLAST, junto con consultas a APIs externas, aportan funcionalidades complementarias para ejecutar flujos completos que incluyen la traducción de secuencias, la búsqueda de homólogos y la comparación de estructuras tridimensionales para facilitar la evaluación del impacto funcional de mutaciones.

Abstract

This project presents a web application for the analysis of genomic data developed to investigate how certain genetic variations may be associated with conditions such as rickets across different populations. Based on a user-provided sequence, the platform allows comparison with reference sequences and structures, identifying possible genetic factors that can explain phenotypic differences.

Through an architecture of Docker-based microservices, coordinated by a backend in .NET and a frontend in React with TypeScript, internal services such as Bioconductor, Biopython and BLAST, along with queries to external APIs, provide complementary functionalities to execute complete analytical workflows, including sequence translation, homology searching and comparison of three-dimensional structures to facilitate the evaluation of the functional impact of mutations.

Agradecimientos

Quiero agradecer a todas las personas que me acompañaron y me apoyaron en este proceso, en el que gran parte de mi atención estuvo puesta acá. A mi hija, mi pareja, mi mamá, mis abuelas, a mis amigos y compañeros de trabajo. A mis compañeros de estudio, con quienes compartimos el esfuerzo y nos ayudamos para avanzar en las materias cuando se hizo difícil. A los docentes de la carrera y a la UNAJ, que me dieron la posibilidad de formarme en la ciudad donde nací.

Índice

Resumen.....**2**
Abstract.....***2***
Agradecimientos.....**3**
Índice.....**4**
Índice de figuras.....**9**
Índice de tablas.....**11**
Sección 1.....**13**
1.1 Introducción.....13
1.2 Motivación del proyecto.....13
Sección 2.....**14**
2.1 Objetivos.....14
2.2 Alcances.....14
2.3 Requerimientos.....14
2.3.1 Requerimientos funcionales.....14
2.3.2 Requerimientos no funcionales.....18
Sección 3: Marco teórico.....**21**
3.1 Contexto biológico.....21
3.1.1 Estructura y función celular.....21
3.1.2 Estructura del ADN y ARN.....21
3.1.3 Ubicación citogenética y molecular.....22
3.1.4 Código genético y expresión génica.....23
3.1.4.1 Transcripción.....23
3.1.4.2 Codón y ARNm.....24
3.1.4.4 Anticodón y ARN de transferencia.....24
3.1.4.5 Traducción.....25
3.1.4.3 Aminoácidos.....25
3.1.4.6 Proteinas.....26
Estructura primaria.....27
Estructura secundaria.....27
Estructura terciaria.....27
Estructura cuaternaria.....27
3.1.4.7 Homologia.....28
3.1.4.8 Variabilidad genética.....28
3.1.5 Vitamina D.....28
3.2 Contexto bioinformático.....28
3.2.1 Formatos de archivos bioinformáticos.....28
3.2.1.1 FASTA (.fasta, .fa).....28
3.2.1.2 PDB (.pdb).....29
3.2.1.3 JSON (JavaScript Object Notation o Notación de Objetos de JavaScript).....29

3.2.2 Fuentes de datos..... 30
3.2.2.1 NCBI (National Center for Biotechnology Information)..... 30
3.2.2.2 RCSB PDB (Research Collaboratory for Structural Bioinformatics Protein Data Bank)..... 30
3.2.2.3 UniProt (Universal Protein Resource)..... 30
3.2.2.4 AlphaFoldDB..... 31
pLDDT(predicted Local Distance Difference Test)..... 31
PAE (Predicted Aligned Error)..... 31
3.2.3 Herramientas de análisis bioinformático..... 31
3.2.3.1 BLAST (Basic Local Alignment Search Tool)..... 31
Alineamiento..... 31
Matrices de sustitución..... 32
Heurística basada en semillas..... 32
Modelo de Karlin–Altschul..... 32
BLASTX..... 33
3.2.3.2 Bioconductor..... 35
Módulo BSgenome..... 35
Módulo Biostrings..... 35
Módulo org.Hs.eg.db..... 35
Módulo TxDb.Hsapiens.UCSC.hg38.knownGene..... 35
3.2.3.3 Biopython..... 35
Módulo Bio.PDB..... 36
3.2.3.4 AlphaFold..... 36
RMSD (root-mean-square deviation)..... 36
Implementación..... 37
3.2.3.5 Neurosnap..... 37
3.2.3.6 3Dmol.js..... 38
3.2.3.7 Plotly..... 38
3.2.4 Entornos de desarrollo..... 38
3.2.4.1 Visual Studio Code..... 38
3.2.4.2 Visual Studio..... 39
3.2.5 Infraestructura y soporte..... 39
3.2.5.1 Docker..... 39
Docker Desktop..... 39
Docker Hub..... 39
3.2.5.2 Arquitectura cliente-servidor..... 39
3.2.5.3 Arquitectura en capas..... 39
Inyección de dependencias (Dependency Injection, DI)..... 40
3.2.5.4 API (Application Programming Interface)..... 40
3.2.5.5 REST (Representational State Transfer)..... 40
3.2.5.6 Middleware de manejo de excepciones..... 40

3.2.5.7 Documentación de APIs..... 40
3.2.5.8 Fetch API..... 41
3.2.5.9 HTTP en SPA (Single Page Application)..... 41
3.2.5.10 HTTPS (HyperText Transfer Protocol Secure)..... 41
3.2.5.11 Navegador..... 41
3.2.5.12 CORS (Cross-Origin Resource Sharing)..... 41
3.2.5.13 Tecnologías y frameworks..... 41
3.2.5.14 Controlador de versiones (VCS)..... 42
3.3 Contexto histórico y científico del caso de estudio..... 42
Sección 4: Diseño de solución..... 43
4.1 Arquitectura..... 43
4.2 Servicio orquestador: .NET..... 44
4.3 Servicios internos y externos..... 45
4.4 Comunicación entre microservicios..... 46
Sección 5: Planificación..... 47
5.1 Tareas a desarrollar..... 47
5.2 Cronograma de trabajo..... 48
Sección 6: Desarrollo e implementación..... 49
6.1 Entorno de ejecución y gestión de contenedores..... 49
6.2 Servicios..... 50
6.2.1 Servicio bioc..... 50
6.2.1.1 Estructura del repositorio..... 50
6.2.1.2 Imagen Docker del servicio..... 51
6.2.1.3 Documentación de APIs..... 51
6.2.1.4 Endpoints bioc..... 52
/autocomplete..... 52
/detail..... 52
/detailfull..... 53
/entrez..... 54
/isentrez..... 55
/sequence_by_range..... 55
/align..... 56
/percent..... 57
6.2.2 Servicio blast..... 58
6.2.2.1 Estructura del repositorio..... 58
6.2.2.2 Imagen Docker del servicio..... 58
6.2.2.3 Documentación de APIs..... 61
6.2.2.4 Endpoints blast..... 61
/blastx..... 61
6.2.3 Servicio biopython..... 63
6.2.3.1 Estructura del repositorio..... 63

6.2.3.2 Imagen Docker del servicio....64
6.2.3.3 Documentación de APIs....65
6.2.3.4 Endpoints biopython....65
/translate....65
/reverse-complement....66
/multipart....67
6.2.4 Servicio presentation....68
6.2.4.1 Patrón de Capas....68
Capa de Presentación....68
Capa de Aplicación....69
Capa de Dominio....70
Capa de Infraestructura....70
6.2.4.2 Control de CORS....71
6.2.4.3 Inyección de dependencias....71
6.2.4.4 Middleware....72
6.2.4.5 Servicios externos....72
NEUROSNAP....72
NCBI E-utilities....73
RCSB PDB....73
UniProtKB API....73
6.2.4.6 Documentación de APIs....73
6.2.4.7 Endpoints backend principal....74
/api/Plumber....74
/api/Blast....76
/api/Folding....77
/api/Public....79
/api/Python....80
6.2.5 Servicio web....82
6.2.5.1 Estructura del repositorio....82
6.2.5.2 Imagen Docker del servicio....83
6.2.5.3 Vistas....84
Vista blastx....85
Vista detail....88
Vista search....92
Vista align....93
6.3 Funcionalidades principales....93
6.3.1 Análisis estructural....93
6.3.1.1 Diagrama de flujo....93
6.3.1.2 Diagrama de procesos....95
Procesar secuencia con blastx....95
Traducción....95

Predicción de estructura con Neurosnap..... 95
Descarga del modelo de referencia..... 95
Alineamiento..... 95
6.3.2 Detalles..... 97
6.4 Metodología de desarrollo..... 99
6.5 Verificación..... 99
6.6 Dificultades..... 99
6.5.1 Interoperabilidad..... 99
6.5.2 Base de datos para BLAST..... 99
6.5.3 Predicción de estructuras de proteínas..... 100
6.5.4 Despliegue en la nube..... 100
6.5.5 Extensiones del navegador..... 100
Sección 7: Conclusiones..... 101
Bibliografía..... 103

Índice de figuras

Figura 1. Célula con núcleo y cromosoma formado por ADN.
Figura 2. Distribución de los nucleótidos con desoxirribosa en una doble cadena de ADN.
Figura 3. Ubicación del gen A2MD
Figura 4. Transcripción y traducción.
Figura 5. Traducción de codón a aminoácido
Figura 6. Codon y anticodon en la traducción.
Figura 7. Estructura del aminoácido.
Figura 8. Enlace peptídico.
Figura 9. Expresión génica.
Figura 10. Cuatro niveles de estructura proteica.
Figura 11. Formato de archivo FASTA.
Figura 12. Formato de archivo PDB.
Figura 13. Formato JSON.
Figura 14. Configuración de la API de Neurosnap.
Figura 15. Dockerfile, imagen y contenedor.
Figura 16. Docker-compose.yml
Figura 17. Diagrama de clases UML
Figura 18. Captura de pantalla de Docker Desktop.
Figura 19. Captura de pantalla de plumber.R
Figura 20. Captura de pantalla del Dockerfile bioc
Figura 21. Captura de pantalla de Swagger (bioc).
Figura 22. Captura de pantalla de blast_api.R
Figura 23. Captura de pantalla del Dockerfile blast.
Figura 24. Captura de pantalla de Swagger (blast).
Figura 25. Captura de pantalla de blastx.R
Figura 26. Captura de pantalla del JSON estandarizado de la respuesta blastx.
Figura 27. Captura de pantalla de api.py
Figura 28. Captura de pantalla del Dockerfile biopython.
Figura 29. Captura de pantalla de Swagger (biopython).
Figura 30. Captura de pantalla de la Solución .NET
Figura 31. Captura de pantalla de la estructura de carpetas del proyecto presentation.
Figura 32. Captura de pantalla de la estructura de carpetas del proyecto application.
Figura 33. Captura de pantalla de la estructura de carpetas del proyecto domain.
Figura 34. Captura de pantalla de la estructura de carpetas del proyecto infraestructure.
Figura 35. Captura de pantalla de la política de CORS en el archivo Program.cs..
Figura 36. Captura de pantalla del registro de servicios y clientes en el contenedor de dependencias (Parte 1).
Figura 37. Captura de pantalla del registro de servicios y clientes al contenedor de dependencias (Parte 2).
Figura 38. Captura de pantalla de la declaración de Middleware en el archivo Program.cs.
Figura 39. Captura de pantalla de Swagger (backend principal)
Figura 40. Captura de pantalla de la estructura de archivos web.

Figura 41. Captura de pantalla del código del Dockerfile web
Figura 42. Captura de pantalla de la Vista Home.
Figura 43. Captura de pantalla de la Navegación.
Figura 44. Captura de pantalla del componente Upload Sequence en la vista blastx.
Figura 45. Captura de pantalla de la secuencia query en la vista blastx.
Figura 46. Captura de pantalla del modal de resultados de blast en la vista blastx.
Figura 47. Captura de pantalla de la Sección para traducir la secuencia a predecir en la vista blastx.
Figura 48. Captura de pantalla de la Notificación para ingresar la API key.
Figura 49. Captura de pantalla del Modal de configuración para ingresar la API Key de Naurosnap.
Figura 50. Captura de pantalla de la Sección para obtener las predicciones de Neurosnap en la vista blastx.
Figura 51. Captura de pantalla del Modal Structure con el Render 3Dmol.js en la vista blastx.
Figura 52 Captura de pantalla del Modal de búsqueda.
Figura 53. Captura de pantalla del detalle en la vista Detail.
Figura 54. Captura de pantalla del detalle completo en la vista Detail.
Figura 55. Captura de pantalla de la secuencia del gen en Detail Full.
Figura 56. Captura de pantalla de las estadísticas de la secuencia en Detail Full.
Figura 57. Captura de pantalla del Modal de estructura según el UniprotID en la vista Detail Full.
Figura 58. Captura de pantalla de la vista de búsqueda de secuencias.
Figura 59. Captura de pantalla de la vista de secuencia reversa y complementaria.
Figura 60. Captura de pantalla de la Vista Align.
Figura 61. Diagrama de flujo de la funcionalidad de Análisis estructural.
Figura 62. Diagrama blastx
Figura 63. Diagrama de proceso de la funcionalidad de Análisis estructural (Parte 1).
Figura 64. Diagrama de proceso de la funcionalidad de Análisis estructural (Parte 2).
Figura 65. Diagrama de flujo de la funcionalidad de Detalle.

Índice de tablas

Tabla 1. RF01
Tabla 2. RF02
Tabla 3. RF03
Tabla 4. RF04
Tabla 5. RF05
Tabla 6. RF06
Tabla 7. RF07
Tabla 8. RF08
Tabla 9. RF09
Tabla 10. RF010
Tabla 11. RNF01
Tabla 12. RNF02
Tabla 13. RNF03
Tabla 14. RNF04
Tabla 15. RNF05
Tabla 16. RNF06
Tabla 17. Constantes globales utilizadas por BLASTX
Tabla 18. Parámetros reportados por BLASTX
Tabla 19. Cronograma de trabajo
Tabla 20. Estructura de archivos bioc
Tabla 21. /autocomplete/
Tabla 22. /detail/
Tabla 22. /detailfull/
Tabla 24. /entrez/
Tabla 25. /isentrez/
Tabla 26. /sequence_by_range/
Tabla 27. /align/
Tabla 28. /percent/
Tabla 29. Estructura de archivos blast
Tabla 30. /blastx/
Tabla 31. Estructura de archivos biopython.
Tabla 32. /translate/
Tabla 33. /reverse-complement/
Tabla 34. /multipart/
Tabla 35. /api/
Tabla 36. /autocomplete/
Tabla 37. /align/
Tabla 38. /detail/
Tabla 39. /percent/
Tabla 40. /sequence/
Tabla 41. /entrez/
Tabla 42. /blastx/

Tabla 43. /init/
Tabla 44. /status/
Tabla 45. /ranks/
Tabla 46. /align/
Tabla 47. /pLDDT/
Tabla 48. /model/
Tabla 49. /summary/
Tabla 50. /model/
Tabla 51. /complement/
Tabla 52. /translate/
Tabla 53. /compare/

Sección 1

1.1 Introducción

El Proyecto Genoma Humano (1990-2003) permitió secuenciar y decodificar el ADN abriendo la posibilidad de identificar mutaciones y variaciones genéticas asociadas a enfermedades observadas fenotípicamente. Por otro lado, en la actualidad existen aplicaciones, tales como *AlphaFold*, capaces de predecir con muy elevada confianza la estructura tridimensional de proteínas en base a su secuencia de aminoácidos mediante predicciones con pLDDT (por las siglas en inglés de *predicted Local Distance Difference Tes*t) mayores al 90%. Dicha estructura es fundamental para que la proteína cumpla su función en la célula. El análisis de estas variaciones podría acercarnos a entender los posibles efectos de las mutaciones y su impacto biológico.

Por esto, desde la Ingeniería Informática, el presente Proyecto Integrador Profesionalizante (PIP) se centra en el diseño y desarrollo de una aplicación web para el análisis de datos genómicos que integre herramientas bioinformáticas en un entorno accesible, escalable y usable para profesionales del área de la bioinformática.

La aplicación permitirá el alineamiento de lecturas con el genoma humano de referencia para identificar variaciones genéticas, e integrará la predicción de estructuras de proteínas para así poder compararlas y facilitar el análisis del impacto funcional de las mutaciones del caso de uso propuesto.

1.2 Motivación del proyecto

La motivación principal de este proyecto surge de la necesidad de una herramienta que aproveche los avances en bioinformática que permitieron estudiar cómo influyen las variaciones genéticas en la salud de las personas, pero que, a la vez, no requiera conocimientos avanzados para usuarios no especializados en programación.

Si bien la aplicación es de uso general, se plantea el estudio de un caso particular que permite definir qué variables, flujos y análisis es necesario incorporar. Dicho caso es el raquitismo históricamente prevalente en las poblaciones no originarias, que colonizaron la Patagonia austral a finales del siglo XIX y principios del siglo XX (especialmente en Punta Arenas en 1848, la ciudad más antigua de la zona), pero ausente en poblaciones originarias, lo cual plantea interrogantes en torno a la genética y la influencia de la vitamina D en la salud, dado que esta última se utiliza para prevenir el raquitismo.

El presente proyecto tiene como propósito aportar una solución informática que unifique distintas herramientas bioinformáticas en un único sistema accesible vía web, utilizando contenedores Docker, APIs y servicios externos. De esta manera, se contribuye desde la informática, al avance de investigaciones en Biología Molecular, enfocándose en el desarrollo de software robusto, escalable y orientado al usuario final.

Finalmente, en lo personal, este proyecto me permite la aplicación e integración de conocimientos adquiridos durante la carrera. Además, el desarrollo de este tipo de soluciones puede facilitar el estudio de casos bioinformáticos e impactar en ámbitos de investigación biomédica.

Sección 2

2.1 Objetivos

- Desarrollar una aplicación web en React Typescript para análisis de datos genómicos.
- Implementar un backend que integre servicios bioinformáticos mediante contenedores Docker.
- Analizar genes relacionados con el metabolismo de la vitamina D.

2.2 Alcances

Si bien el proyecto está orientado al caso de estudio, la aplicación podrá utilizarse para otros análisis genómicos. Principalmente se incluirán las siguientes funciones:

- La traducción de secuencias de nucleótidos a secuencia de aminoácidos.
- La comparación de secuencias de entrada con proteínas de referencia en bases de datos curadas.
- La predicción de la estructura tridimensional de proteínas a través de herramientas externas.
- La visualización y comparación de la estructura tridimensional de proteínas.

Además, el sistema contará con las siguientes funciones utilitarias:

- El detalle de genes (ubicación citogenética, descripción, rango y hebra, nucleótidos que lo componen, histogramas de regiones CpG),
- La obtención de secuencias por gen o por rango.
- La obtención de secuencias complemento y reversa de una secuencia de entrada.
- El alineamiento de secuencias de nucleótidos y su visualización mediante dot plots.

Se plantean como funciones opcionales, la generación de reportes y la descarga de archivos.

2.3 Requerimientos

A continuación se definen los requerimientos del sistema, tanto funcionales como no funcionales.

2.3.1 Requerimientos funcionales

Se describen las acciones que el sistema debe realizar para cumplir los objetivos:

Tabla 1. RF01

Identificación	RF01
Nombre	Carga de archivos locales.
Características	Permite al usuario ingresar archivos de secuencias desde su equipo.
Descripción	El sistema deberá permitir la carga de archivos locales que contengan secuencias de ADN o proteínas en formatos compatibles (.fasta, .pdb, etc.).
Prioridad	Alta.

Tabla 2. RF02

Identificación	RF02
Nombre	Consulta en bases de datos públicas.
Características	Acceso a información biológica en repositorios externos.
Descripción	El sistema deberá consultar bases de datos públicas para obtener información complementaria sobre genes, proteínas o estructuras.
Prioridad	Alta.

Tabla 3. RF03

Identificación	RF03
Nombre	Procesamiento y análisis de secuencias.
Características	Incluye herramientas bioinformáticas integradas.
Descripción	El sistema deberá procesar y analizar secuencias mediante funciones como traducción de nucleótidos a aminoácidos, alineamiento y predicción estructural, integrando servicios correspondientes en los contenedores cuando corresponda.
Prioridad	Alta.

Tabla 4. RF04

Identificación	RF04
Nombre	Visualización de resultados.
Características	Representación gráfica y tabular de los análisis realizados.
Descripción	El sistema deberá mostrar los resultados obtenidos mediante gráficos, tablas y representaciones tridimensionales de proteínas para facilitar la interpretación del análisis desde el frontend.
Prioridad	Alta.

Tabla 5. RF05

Identificación	RF05
Nombre	Integración de servicios dockerizados.
Características	Interoperabilidad entre módulos de análisis bioinformático.
Descripción	El sistema deberá integrar y ejecutar herramientas bioinformáticas en contenedores asegurando la modularidad e interoperabilidad en el intercambio de datos normalizados mediante DTOs REST.
Prioridad	Alta.

Tabla 6. RF06

Identificación	RF06
Nombre	Normalización de datos.
Características	Estandarización de respuestas REST.
Descripción	El backend .NET deberá recibir los resultados de los servicios externos, procesarlos y devolver un DTO normalizado que pueda ser interpretado por la interfaz web.
Prioridad	Alta.

Tabla 7. RF07

Identificación	RF07
Nombre	Generación y descarga de archivos.
Características	Exportación de resultados.
Descripción	El sistema deberá permitir la generación y descarga de archivos de alineamiento en formato estándar (.pdb).
Prioridad	Media.

Tabla 8. RF08

Identificación	RF08
Nombre	Manejo centralizado de errores.
Características	Gestión uniforme de excepciones provenientes de servicios externos.
Descripción	El sistema deberá capturar las excepciones generadas por los servicios Python y R mediante un middleware en .NET, normalizando los mensajes de error y devolviéndolos al frontend con un formato estándar.
Prioridad	Alta.

Tabla 9. RF09

Identificación	RF09
Nombre	Notificaciones al usuario.
Características	Comunicación visual del estado del sistema.
Descripción	El sistema deberá notificar al usuario los estados de carga, éxito o error mediante elementos visuales (toasts o loaders) que se activen durante la interacción con los servicios.
Prioridad	Media.

Tabla 10. RF10

Identificación	RF10
Nombre	Interfaz interactiva.
Características	Componentes UI.
Descripción	El sistema deberá ofrecer una interfaz web que incluya barra de navegación, buscador con autocompletado, vistas de detalle, visualización 3D de proteínas, modales, tablas y carga de archivos.
Prioridad	Media.

2.3.2 Requerimientos no funcionales

Se establecen las condiciones de calidad, rendimiento y diseño bajo las cuales debe operar.

Tabla 11. RNF01

Identificación	RNF01
Nombre	Exposición del backend como API REST.
Características	Comunicación estándar entre frontend y backend.
Descripción	El backend desarrollado en .NET deberá exponer sus funcionalidades a través de una API RESTful accesible por la aplicación React.
Prioridad	Alta.

Tabla 12. RNF02

Identificación	RNF02
Nombre	Orquestación de microservicios.
Características	Ejecución coordinada de contenedores.
Descripción	El sistema deberá utilizar Docker Compose para orquestar los contenedores correspondientes a los servicios en .NET, Python y R.
Prioridad	Alta.

Tabla 13. RNF03

Identificación	RNF03
Nombre	Escalabilidad y modularidad.
Características	Capacidad de agregar o actualizar servicios sin afectar al sistema principal.
Descripción	El diseño deberá permitir el escalado horizontal y la integración de nuevas herramientas con mínima reconfiguración.
Prioridad	Media.

Tabla 14. RNF04

Identificación	RNF04
Nombre	Usabilidad.
Características	Interfaz intuitiva y clara.
Descripción	La interfaz React deberá presentar la información de forma comprensible, con elementos gráficos que faciliten la interpretación de resultados a usuarios no técnicos.
Prioridad	Alta.

Tabla 15. RNF05

Identificación	RNF05
Nombre	Tolerancia a fallos y manejo de errores.
Características	Robustez ante fallas de servicios.
Descripción	El sistema deberá garantizar que las fallas o errores en servicios externos no interrumpan la ejecución normal, manteniendo la disponibilidad del backend y notificando adecuadamente al usuario.
Prioridad	Alta.

Tabla 16. RNF06

Identificación	RNF06
Nombre	Experiencia de usuario.
Características	Fluidez e interacción con el usuario.
Descripción	La interfaz React deberá proporcionar una experiencia de usuario fluida, con transiciones, retroalimentación inmediata y componentes accesibles que faciliten el uso del sistema sin conocimientos técnicos avanzados.
Prioridad	Media.

Sección 3: Marco teórico

3.1 Contexto biológico

3.1.1 Estructura y función celular

Una célula es la mínima unidad estructural y funcional básica de los organismos vivos. En una célula típica se distinguen el núcleo, que contiene la información genética, y el citoplasma, donde ocurren la mayoría de los procesos metabólicos; ambos separados por membranas (nuclear o celular) que regulan el intercambio de sustancias.

El núcleo de cada célula la controla para que crezca y madure, se replique o muera. En él se encuentran aproximadamente 30.000 genes diferentes que pueden ser procesados por la célula en más de una forma, para dar origen a versiones alternativas de una proteína, siendo capaces de producir al menos 100.000 proteínas tanto estructurales (las que dan estructura a los tejidos), como enzimáticas (aquellas que catalizan reacciones químicas).

Cada gen está compuesto por macromoléculas de ácido desoxirribonucleico (ADN) que controlan la formación del ácido ribonucleico (ARN) que se dispersa por toda la célula para controlar la formación de una proteína específica. Dentro de las células, el ADN está organizado en estructuras llamadas cromosomas. Dentro de los cromosomas, el ADN se compacta enrollándose en complejos de ocho proteínas llamadas histonas para formar nucleosomas, las unidades básicas de empaquetamiento del ADN en eucariotas.

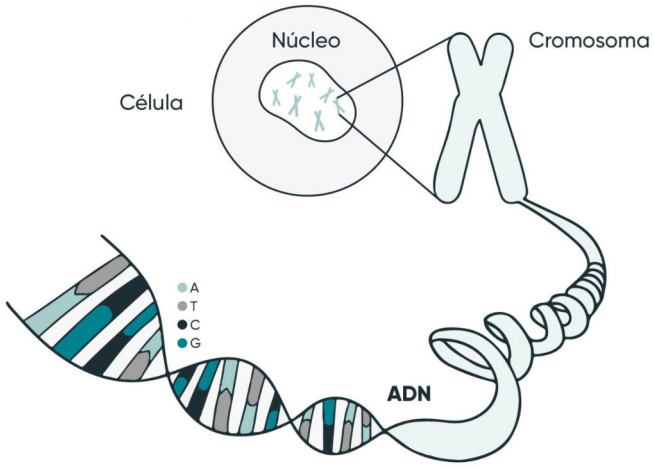

Figura 1. *Célula con núcleo y cromosoma formado por ADN.*

Nota. Adaptado de "Qué son los genes, cómo se expresan", de ADN-Institut (s. f.), https://www.adninstitut.com/que-son-los-genes-como-se-expresan-n-66-es

3.1.2 Estructura del ADN y ARN

Los compuestos químicos básicos implicados en la formación de ADN son tres: el ácido fosfórico (P), el azúcar desoxirribosa (D) y una base nitrogenada.

La combinación de una molécula de ácido fosfórico (P), una molécula de desoxirribosa (D) y una de las cuatro bases nitrogenadas que puede ser purina (Adenina y Guanina) o pirimidina (Timina o Citosina) permite la formación de cuatro nucleótidos: los ácidos desoxiadenílico, desoxitimidílico, desoxiguanílico y desoxicitidílico.

El esqueleto de cada hebra de ADN está compuesto por moléculas de ácido fosfórico y desoxirribosa alternados, que le dan soporte para la molécula de ADN. Mientras las bases nitrogenadas se conectan entre sí mediante enlaces débiles de hidrógeno. Siempre, cada base purínica A de una hebra se une con una base pirimidínica T de la otra.Y cada base purínica G se une con una base pirimidínica de C; dando lugar a pares de bases complementarios: AT, TA, CG, GC.

(El ARN usa casi los mismos bloques básicos excepto que no usa el azúcar desoxirribosa sino el azúcar ribosa y la timina se reemplaza por otra pirimidina, el uracilo.)

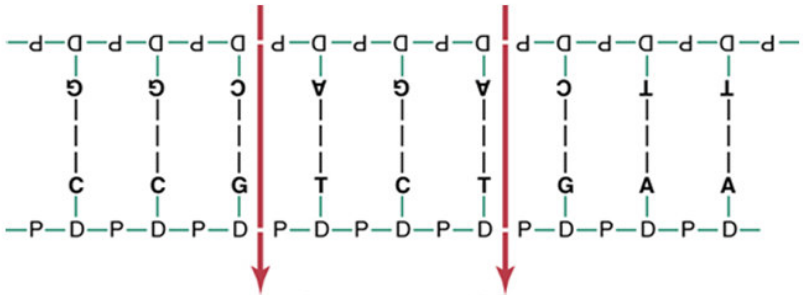

Figura 2. *Distribución de los nucleótidos con desoxirribosa en una doble cadena de ADN.*

Nota. *Tomado de "Tratado de fisiología médica" (14.ª ed.) de A. C. Guyton y J. E. Hall, 2021, Elsevier.*

La secuencia ordenada de nucleótidos no solo determina la información genética, sino que también permite ubicar cada gen dentro del genoma.

3.1.3 Ubicación citogenética y molecular

La posición de un gen dentro de un cromosoma puede describirse a través de dos tipos de mapas:

La ubicación citogenética se obtiene a partir de un patrón de bandas (que se observa cuando los cromosomas se tiñen) que permiten identificar regiones, brazos y subregiones cromosómicas.

Por otro lado, la ubicación molecular, se basa en la secuencia de bloques de construcción de ADN que forman el cromosoma, describiendo la posición exacta del gen mediante la numeración de las bases del ADN; lo cual brinda una referencia precisa basada en la secuencia.

Por ejemplo, en el gen A2MD, la ubicación citogenética 12q13.31 representa la posición 13.31 en el brazo largo (q) del cromosoma 12. Y su ubicación molecular indica que su rango va desde la posición 9.067.664 a la 9.116.229, con una longitud total de 48.566 nucleótidos.

Cytogenetic Location	12p13.31
Location 0	chr12: 9067664 to 9116229 (lenght: 48566) | Strand 3′ → 5′ (-)

Figura 3. Ubicación del gen A2MD

Nota. Captura de la aplicación en desarrollo.

3.1.4 Código genético y expresión génica

Se conoce como expresión génica el proceso que va desde la transcripción del código genético en el núcleo (la formación de ARNm) hasta la traducción del código del ARN (de nucleótidos a aminoácidos) y la formación de proteínas en el citoplasma celular.

Como el ADN se encuentra en el núcleo de la célula pero la mayoría de las funciones de la célula se realizan en el citoplasma, se necesita la formación de otro tipo de ácido nucleico encargado de transportar el código hasta él: el ARN.

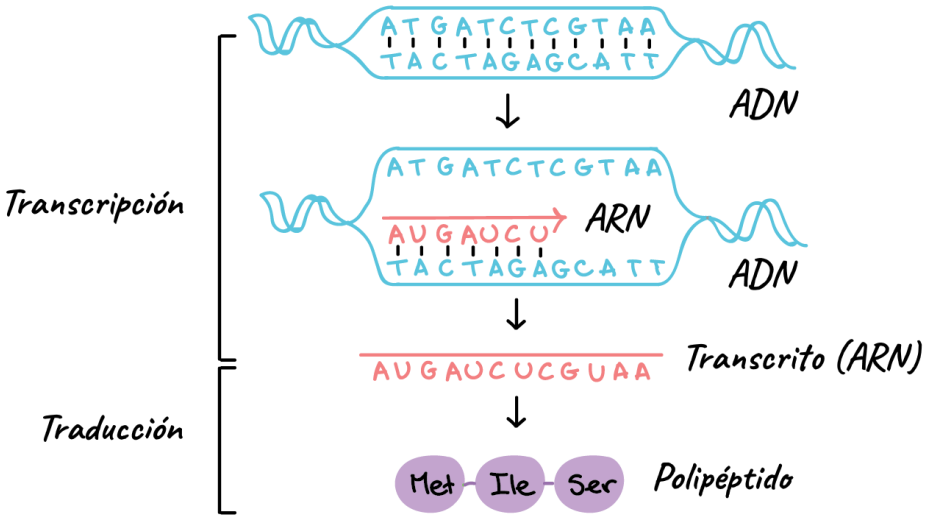

Figura 4. Transcripción y traducción.

Nota. Adaptado de Khan Academy (s.f.), https://es.khanacademy.org/science/ap-biology/gene-expression-and-regulation/transcription-and-rna-processing/a/overview-of-transcription

3.1.4.1 Transcripción

La enzima polimerasa reconoce en el ADN una región llamada promotor, que está situada antes del gen que será copiado, y se une a él para iniciar la transcripción. Para esto es necesario que las dos hebras de la molécula de ADN, que ya contiene la información para fabricar proteínas, se separen temporalmente durante la síntesis de ARN dejando expuestas

las bases nitrogenadas que se utilizarán como plantilla para transferir el código genético al ARN que avanza por la hebra molde incorporando nucleótidos de ARN complementarios para formar la nueva cadena. Cuando la enzima alcanza el extremo del gen y se separa del ADN, la cadena de ARN se desprende y se libera en el nucleoplasma y el ADN, dada su afinidad para volver a unirse con su propia cadena complementaria, recupera su estructura.

3.1.4.2 Codón y ARNm

El ARN mensajero (ARNm) está formado por cadenas que se encuentran en el citoplasma, compuestas por cientos a miles de nucleótidos formando tripletes complementarios a los tripletes de los genes en el ADN.

Cada triplete de bases o codón determina un aminoácido específico durante la síntesis de proteínas en el citoplasma y, si bien hay una combinación de 4 bases tomadas de a 3 (43 combinaciones), como se observa en la Figura 5, uno o más codones de ARN producen los 20 aminoácidos más frecuentes de las moléculas proteicas. También cuenta con un codón de iniciación (CI) que representa la señal de iniciar la fabricación y tres codones de terminación (CT) que representan la señal de detener la fabricación de la proteína.

	U				C				A				G			
U	U	U	U	Phe	U	C	U	Ser	U	A	U	Tyr	U	G	U	Cys
	U	U	C	Phe	U	C	C	Ser	U	A	C	Tyr	U	G	C	Cys
	U	U	A	Leu	U	C	A	Ser	U	A	A	End	U	G	A	End
	U	U	G	Leu	U	C	G	Ser	U	A	G	End	U	G	G	Trp
C	C	U	U	Leu	C	C	U	Pro	C	A	U	His	C	G	U	Arg
	C	U	C	Leu	C	C	C	Pro	C	A	C	His	C	G	C	Arg
	C	U	A	Leu	C	C	A	Pro	C	A	A	Gin	C	G	A	Arg
	C	U	G	Leu	C	C	G	Pro	C	A	G	Gin	C	G	G	Arg
A	A	U	U	Ile	A	C	U	Thr	A	A	U	Asn	A	G	U	Ser
	A	U	C	Ile	A	C	C	Thr	A	A	C	Asn	A	G	C	Ser
	A	U	A	Ile	A	C	A	Thr	A	A	A	Lys	A	G	A	Arg
	A	U	G	Met	A	C	G	The	A	A	G	Lys	A	G	G	Arg
G	G	U	U	Val	G	C	U	Ala	G	A	U	Asp	G	G	U	Gly
	G	U	C	Val	G	C	C	Ala	G	A	C	Asp	G	G	C	Gly
	G	U	A	Val	G	C	A	Ala	G	A	A	Glu	G	G	A	Gly
	G	U	G	Val	G	C	G	Ala	G	A	G	Glu	G	G	G	Gly

Figura 5. Traducción de codón a aminoácido

3.1.4.4 Anticodón y ARN de transferencia

El ARN de transferencia (ARNt) es una molécula encargada de transportar un aminoácido específico hacia el ribosoma durante la traducción. Cada tipo de ARNt reconoce un codón particular del ARNm gracias al anticodón, un triplete de nucleótidos presente en una región del ARNt. Y ambos se unen mediante enlaces débiles posicionando y liberando al aminoácido en el lugar apropiado de la cadena de proteína en formación.

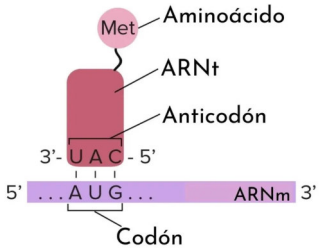

Figura 6. Codon y anticodon en la traducción.

Nota. Adaptado de Lifeder (s.f.), "Anticodón" https://www.lifeder.com/anticodon/

3.1.4.5 Traducción

Es el proceso mediante el cual el ARNm se desplaza por el ribosoma y, con la participación del ARNt, mediante la formación de enlaces peptídicos, se va incorporando a la secuencia cada aminoácido en el orden indicado por los codones para formar la molécula proteica.

3.1.4.3 Aminoácidos

Los aminoácidos[1] son bloques de las proteínas que se encuentran en todas las células vivas y, aunque cumplen otras funciones en las células, su papel más importante es como constituyentes de las proteínas. Todos tienen la misma estructura básica, un carbono α unido a cuatro grupos: un hidrógeno, un grupo carboxilo ($-COOH$), un grupo amino ($-NH_2$) y un grupo o cadena lateral R que lo caracteriza definiendo su función.

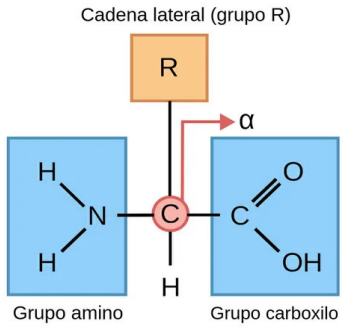

Figura 7. Estructura de un aminoácido.

Nota. Tomado de "Aminoácidos", de Labster (s. f.), https://theory.labster.com/es/amino-acids/

Los aminoácidos están unidos entre sí por enlaces peptídicos, en los que el grupo carboxilo de un aminoácido se une al grupo amino del siguiente, liberando una molécula de agua. En una proteína típica, compuesta por muchos aminoácidos (polipéptido), el extremo del

[1] Aminoacidos: http://biochem.science.oregonstate.edu/content/biochemistry-free-and-easy

péptido que tiene un grupo amino libre se denomina extremo N-terminal, mientras que el extremo con el carboxilo libre se denomina extremo C-terminal (para carboxilo).

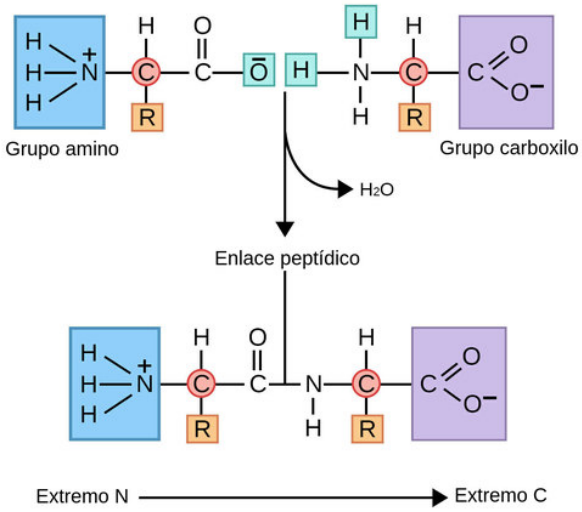

Figura 8. Enlace peptídico.

Nota. Tomado de "Aminoácidos", de Labster (s. f.), https://theory.labster.com/es/amino-acids/

3.1.4.6 Proteinas

Las proteínas[2] se elaboran uniendo aminoácidos. Cada una tiene su secuencia característica y única que se refleja en su estructura y que, a su vez, determina su función. Es por esto que mutaciones en la secuencia de aminoácidos pueden afectar dicha función.

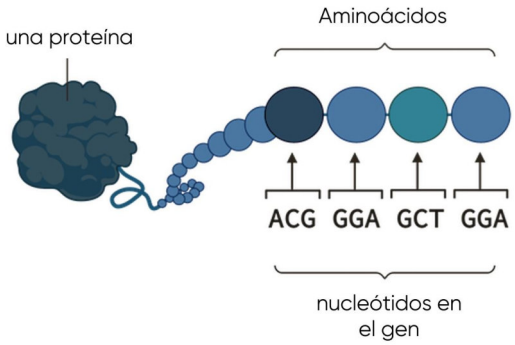

Figura 9. Expresión génica.

Nota. Tomado de "Qué son los genes, cómo se expresan", de ADN-Institut (s. f.), https://www.adninstitut.com/que-son-los-genes-como-se-expresan-n-66-es

Para calcular la diversidad de proteínas que se pueden hacer utilizando los 20 aminoácidos diferentes podemos considerar que un dipéptido (la unión de dos aminoácidos) nos da $20^2 = 400$ combinaciones diferentes.

[2] Proteinas: http://biochem.science.oregonstate.edu/content/biochemistry-free-and-easy

Estructura primaria

La estructura primaria, dada por la secuencia de aminoácidos, determina la conformación de una proteína y es originada por la región o secuencia de ADN que la codifica en el genoma. La región codificante especifica la secuencia de aminoácidos. El orden en que estos aminoácidos se unen en la síntesis de proteínas comienza a definir interacciones para establecer estructuras secundarias y terciarias.

Estructura secundaria

Son patrones locales de plegamiento que adopta la cadena polipeptídica debido a enlaces de hidrógeno entre los grupos del esqueleto peptídico. Pueden ser hélice α *(alpha)* u hoja β *(beta)*. Aunque también existen otras conformaciones como giros y bucles.

Estructura terciaria

Es la conformación tridimensional completa de una cadena polipeptídica que surge de interacciones entre las cadenas laterales (grupos R) de cada aminoácido, que define la forma final y la función.

Estructura cuaternaria

Es el ensamblaje de dos o más cadenas polipeptídicas o proteínas.

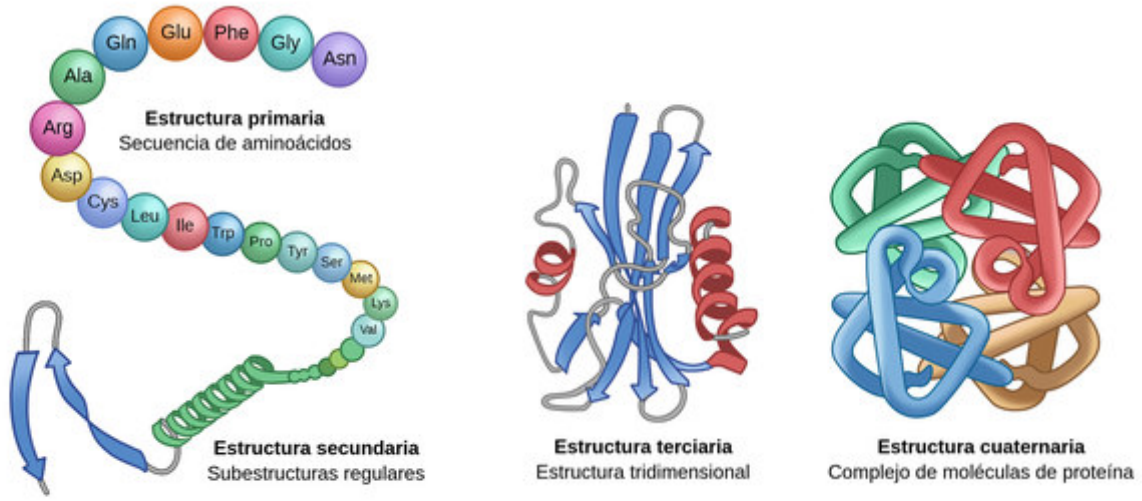

Figura 10. Cuatro niveles de estructura proteica.

Nota. Tomado de "Estructura proteica", de Labster (s. f.), https://theory.labster.com/es/protein-structure/

3.1.4.7 Homologia

La homología es un concepto cualitativo: dos secuencias son homólogas o no lo son. Con frecuencia se confunde con la similitud, que sí es cuantitativa y describe el porcentaje de coincidencia entre residuos.

En la práctica, los niveles de similitud obtenidos mediante alineamientos de secuencia permiten inferir si existe o no homología cuando la similitud obtenida es significativamente mayor a la similitud esperable por azar, es por esto que las herramientas de bioinformática la utilizan para identificar relaciones homólogas entre proteínas. Esto es importante, ya que la homología de proteínas implica que ambas proceden de genes homólogos y, en consecuencia, tienden a conservar elementos clave de estructura tridimensional, que le dan su función biológica.

3.1.4.8 Variabilidad genética

Cuando hablamos de genotipo nos referimos a la composición genética mientras que al hablar de fenotipo se hace referencia a los rasgos observables que resultan de la interacción entre su genotipo y los factores ambientales. El genotipo de una población admite cierta variabilidad o diferencias en el material genético, que pueden clasificarse según su impacto funcional: cuando la alteración no modifica la estructura ni la tarea de la proteína resultante, se considera un polimorfismo. Pero cuando la variación afecta la función proteica, se clasifica como una mutación funcional y puede producir cambios fenotípicos observables.

3.1.5 Vitamina D

La vitamina D pertenece a una familia de compuestos derivados del colesterol. Su forma más relevante en humanos es la vitamina D_3 o colecalciferol que se obtiene de dos modos mediante los rayos UV que convierten el 7-dehidrocolesterol de la piel en colecalciferol o por la ingesta de alimentos o suplementos.

El raquitismo es una enfermedad de la infancia causada por la deficiencia de esta vitamina, que produce la disminución de la absorción intestinal de calcio y fosfato.

3.2 Contexto bioinformático

3.2.1 Formatos de archivos bioinformáticos

3.2.1.1 FASTA (.fasta, .fa)

Es un formato estándar para almacenar secuencias de ADN, ARN o proteínas en texto plano. Cada secuencia comienza con una línea de encabezado con “>” seguida de una descripción y luego la secuencia de residuos de la molécula.

```
>MCHU - Calmodulin - Human, rabbit, bovine, rat, and chicken
MADQLTEEQIAEFKEAFSLFDKDGDGTITTKELGTVMRSLGQNPTEAELQDMINEVDADGNGTID
FPEFLTMMARKMKDTDSEEEIREAFRVFDKDGNGYISAAELRHVMTNLGEKLTDEEVDEMIREA
DIDGDGQVNYEEFVQMMTAK*
```

Figura 11. Formato de archivo FASTA

3.2.1.2 PDB (.pdb)

Es un formato estructural de proteínas mantenido por la Protein Data Bank (RCSB PDB). El fichero PDB describe la estructura tridimensional de la molécula a través de las coordenadas $\{x, y, z\}$ de sus átomos pesados. En una primera parte, denominada *HEADER* contiene información complementaria y las líneas que comienzan con *ATOM* contienen la información estructural. Si existen grupos químicos no proteicos, se escriben en líneas *HETATM* y también pueden haber líneas *ANISOU* con información sobre la flexibilidad de la molécula. Este formato permite almacenar modelos estructurales.

```
ATOM      1  N   MET A   1      28.801 -23.541 -32.151  1.00 35.22           N
ATOM      2  CA  MET A   1      28.628 -22.980 -33.498  1.00 35.22           C
ATOM      3  C   MET A   1      27.576 -21.876 -33.479  1.00 35.22           C
ATOM      4  CB  MET A   1      28.255 -24.060 -34.496  1.00 35.22           C
ATOM      5  O   MET A   1      26.416 -22.128 -33.127  1.00 35.22           O
ATOM      6  CG  MET A   1      29.433 -24.880 -34.990  1.00 35.22           C
ATOM      7  SD  MET A   1      29.159 -25.581 -36.653  1.00 35.22           S
```

Figura 12. Formato de archivo PDB

3.2.1.3 JSON (*JavaScript Object Notation* o Notación de Objetos de JavaScript)

JSON es un formato de texto independiente del lenguaje, legible y rápido de procesar utilizado para el intercambio de datos en estructuras jerárquicas.


```
{
  "code": 200,
  "message": "Ok",
  "data": {
    "entrez": "1717",
    "symbol": "DHCR7",
    "genename": "7-dehydrocholesterol reductase",
    "genetype": "protein-coding",
    "alias": [
      "SLOS",
      "DHCR7"
    ]
  }
}
```


Figura 13. Formato JSON

3.2.2 Fuentes de datos

En bioinformática, las fuentes de datos pueden clasificarse en experimentales y computacionales. Las fuentes de datos experimentales contienen datos obtenidos a través de técnicas de laboratorio y reflejan las mediciones reales de una molécula real.

Las fuentes de datos computacionales almacenan datos generados por modelos predictivos o simulaciones en las que no interviene una medición física directa. Están basadas

en algoritmos, estadística o aprendizaje profundo que les permite predecir las coordenadas de una proteína a partir de su secuencia de aminoácidos.

3.2.2.1 NCBI (*National Center for Biotechnology Information*)

El NCBI[3] depende del National Institute of Health (NIH[4]), un organismo del gobierno de Estados Unidos, mantiene las siguientes bases de datos:

- GenBank: una base de datos de secuencias de ADN y ARN depositadas por la comunidad científica.
- RefSeq: secuencias estandarizadas, anotadas y curadas por NCBI.
- PubMed: es un motor de búsqueda de literatura científica biomédica.
- BLAST: es un conjunto de herramientas de alineamiento local de secuencias contra bases de datos de nucleótidos o proteínas.

3.2.2.2 RCSB PDB *(Research Collaboratory for Structural Bioinformatics Protein Data Bank)*

El Protein Data Bank[5] (PDB) es un archivo global de estructuras tridimensionales de macromoléculas almacenado en tres centros mundiales coordinados por wwPDB: RCSB PDB (Estados Unidos), PDBe (Europa) y PDBj (Japón).
Esta base de datos permite descargar tanto estructuras experimentales como modelos de estructura calculada o CSM *(Computed Structure Models)*.

3.2.2.3 UniProt *(Universal Protein Resource)*

UniProt[6] es un repositorio central de datos sobre proteínas que contiene anotaciones funcionales, características estructurales, etc. Engloba:

- UniProtKB (UniProt Knowledgebase) es la base de datos central de proteínas. Dentro de ella está la Swiss-Prot, que contiene proteínas revisadas manualmente.
- UniRef *(UniProt Reference Clusters)* proporciona conjuntos de secuencias agrupadas por identidad (100%, 90%, 50%).
- UniParc (UniProt Archive) almacena todas las secuencias proteicas únicas conocidas, sin anotaciones, garantizando trazabilidad histórica.

3.2.2.4 AlphaFoldDB

AlphaFoldDB[7] es una base de datos de modelos estructurales computacionales generados por *AlphaFold*. Estos modelos alcanzan una precisión cercana a la de métodos experimentales, lo que permite complementar al PDB cuando no existe estructura experimental para la secuencia. Cada entrada incluye la secuencia de la proteína (que proviene de UniProt), las coordenadas atómicas predichas en formato PDB, la métrica de confianza *pLDDT* para cada residuo y la matriz de error de pares PAE.

---

[3] NCBI: https://www.ncbi.nlm.nih.gov/
[4] NIH: https://www.nih.gov/
[5] RCSB PDB: https://www.rcsb.org/
[6] UniProt: https://www.uniprot.org/
[7] AlphafoldDb: https://alphafold.ebi.ac.uk/

pLDDT*(predicted Local Distance Difference Test)*

Es un indicador de confianza local de la predicción asignado a cada residuo. Evalúa la exactitud local de los átomos Cα de cada residuo de la proteína. Usa una escala de 0 a 100, donde las puntuaciones más altas indican mayor confianza o predicciones más precisas. La puntuación pLDDT[8] puede variar a lo largo de la secuencia, por lo que en rango se clasifica en cuatro categorías de confianza:

- pLDDT ≥ 90: estructura muy confiable, de alta precisión local.
- 70 ≤ pLDDT < 90: buena confianza, es una estructura correcta.
- 50 ≤ pLDDT < 70: baja confianza, con posibles regiones flexibles o desordenadas.
- pLDDT < 50: probablemente una predicción poco confiable.

PAE *(Predicted Aligned Error)*

Esta métrica estima el error esperado en la posición relativa entre pares de residuos y representa cuánto podría desviarse la posición de un residuo respecto de otro al superponerlos.

Se mide en angstroms (Å) y se presenta como una matriz $N \times N$, para N residuos.

Los valores bajos muestran una relación geométrica confiable entre los dos residuos. Mientras que valores altos representan incertidumbre.

3.2.3 Herramientas de análisis bioinformático

3.2.3.1 BLAST *(Basic Local Alignment Search Tool)*

BLAST[9] es un algoritmo y programa de alineamiento de secuencias desarrollado por el NIH. Puede usarse desde el servidor del NCBI pero también está disponible para ser instalado localmente. Implementa un algoritmo de alineamiento local para encontrar regiones similares entre dos secuencias, sin requerir que se alineen completas, y medir esa similitud para encontrar probables genes homólogos e inferir su función.

Alineamiento

La idea de los algoritmos de alineamiento es maximizar la puntuación de la similitud entre dos secuencias. Cuando se comparan estas secuencias se utilizan *"operaciones de edición"* (deleción, inserción y alteración) para modificar una secuencia y que se parezca más a la otra. Cada operación tiene un costo que disminuye la puntuación del alineamiento, por lo que la mejor alineación será aquella que requiera la menor cantidad de modificaciones.

Matrices de sustitución

Como para la biología molecular algunos cambios son más probables que otros, se desarrollaron matrices de sustitución como PAM *(Point Accepted Mutation)* y BLOSUM *(BLOcks of Amino Acid SUbstitution)*, que cuantifican la probabilidad de que un aminoácido sea reemplazado por otro. Asignando puntajes en función de qué tan aceptada es cada sustitución.

---

[8] pLDDT: https://www.ebi.ac.uk/training/online/courses/alphafold/inputs-and-outputs/evaluating-alphafolds-predicted-structures-using-confidence-scores/plddt-understanding-local-confidence/

[9] BLAST: https://academia-lab.com/enciclopedia/blast-biotecnologia/

Heurística basada en semillas

BLAST no implementa el algoritmo de alineamiento local óptimo de Smith-Waterman (un algoritmo preciso pero costoso de comparación de a pares). Sin embargo, toma tres de sus ideas fundamentales:

- En primer lugar, la búsqueda de *High Scoring Pairs* (HSP), que trata de localizar un segmento local con una alta puntuación sin gaps. Una región central de similitud entre la secuencia de consulta y una secuencia de la base de datos.
- En segundo lugar, el uso de matrices de sustitución (BLOSUM62) para evitar interpretar como equivalente cualquier sustitución.
- Por último, la extensión local de coincidencias significativas, que toma el HSP y lo extiende a izquierda y derecha mientras la puntuación acumulada aumenta para evitar recorrer todas las combinaciones posibles.

Esto da como resultado un método aproximado y rápido, pero que no garantiza una solución óptima, por lo que es necesaria la evaluación de la significancia estadística para evaluar si un alineamiento altamente puntuado es producto del azar o representa una similitud biológicamente significativa.

Modelo de Karlin–Altschul.

El modelo utilizado por BLAST para evaluar la significancia estadística de las alineaciones de secuencias locales en biología molecular es el desarrollado por Samuel Karlin y Stephen Altschul en 1990. En él se define la distribución esperada de puntuaciones de alineamientos locales entre dos secuencias generadas de manera aleatoria bajo un conjunto de probabilidades de aparición de aminoácidos y una matriz de sustitución determinada.

El *e-value* indica la probabilidad de obtener este alineamiento por azar y se calcula con la siguiente ecuación:

$$E = K \cdot m \cdot n \cdot e^{(-\lambda S)}$$

donde:

- E es el *e-value* esperado de alineamientos con puntuación mayor o igual a S que se esperarían por azar.
- m es la longitud de la secuencia de consulta.
- n es la longitud de la secuencia de la base de datos.
- S es la puntuación cruda *(raw score)* del alineamiento.
- λ y K son constantes en función de la matriz de sustitución y las probabilidades de cada residuo.

Por ejemplo, si se calcula el alineamiento entre dos secuencias y me da un valor S, el cálculo del modelo de Karlin-Altschul me devuelve el número esperado E de alineamientos con una puntuación mayor a S. Si obtengo un valor esperado bajo significa que hay muy pocas secuencias mayor puntuación, aumentando la probabilidad de que la similitud no sea por azar.

BLASTX

BLASTX utiliza como entrada una secuencia de nucleótidos que se traduce en las seis posibles lecturas de aminoácidos (tres en sentido directo y tres en sentido inverso) y luego compara cada traducción contra una base de datos de proteínas y, para cada alineamiento exitoso,

devuelve las secuencias similares (hits) con métricas para valorar los resultados y una serie de constantes asociadas al modelo estadístico de Karlin–Altschul. Si bien, estos valores no representan características biológicas del hit, son necesarios para calcular las medidas estadísticas que le permitirán al usuario realizar un análisis sobre la significancia del alineamiento (si la similitud es azarosa).

En la Tabla 17 se detallan las constantes globales más relevantes utilizadas por el modelo.

Tabla 17. Constantes globales utilizadas por BLASTX

Campo	Descripción
db_len	Longitud total de la base de datos utilizada en la búsqueda. Se expresa como la suma de las longitudes de todas las secuencias que la componen y es utilizada para normalizar el *E-value*.
db_num	Cantidad de secuencias presentes en la base de datos consultada.
eff_space	El espacio de búsqueda efectivo *(Effective Search Space)* permite calcular la probabilidad de alineamientos aleatorios. Deriva de la longitud de la secuencia consulta y del tamaño total de la base de datos.
entropy	Medida de entropía asociada al modelo probabilístico que describe la variabilidad esperada entre secuencias no relacionadas.
hsp_len	Longitud mínima requerida para un *High-Scoring Segment Pair* (HSP) inicial. Afecta la sensibilidad del algoritmo en la detección de segmentos candidatos.
lambda (λ)	Parámetro fundamental del modelo de *Karlin–Altschul* que relaciona el *raw score* con el *bit score.* Representa la tasa con la que disminuye la probabilidad de observar alineamientos con *scores* altos por azar.
kappa (K)	Constante asociada a la densidad esperada de alineamientos aleatorios. Junto con λ, permite calcular el *E-value* mediante la ecuación estándar del modelo estadístico.

En la Tabla 18 se describen brevemente los parámetros derivados del modelo estadístico y de la alineación, que BLASTX reporta en cada *hit*.

Tabla 18. Parámetros reportados por BLASTX

Campo	Descripción
Hit	Identificador del alineamiento dentro del conjunto de resultados de BLAST.
Len	Longitud del alineamiento entre la secuencia de consulta y la secuencia del *hit*.
Bitscore	Puntuación **normalizada** del alineamiento (a medida que aumenta, el resultado es más significativo es el resultado).

Score	Puntuación **cruda** del alineamiento basada en la matriz de sustitución.
E-value	El valor esperado indica la probabilidad de obtener este alineamiento por azar. Valores bajos representan coincidencias biológicamente significativas.
Identity	Porcentaje de posiciones idénticas entre ambas secuencias en el alineamiento.
Positive	Porcentaje de posiciones donde los aminoácidos son similares (no idénticos), pero considerados *"positivos"* según la matriz.
Name	Nombre o descripción de la proteína o secuencia del *hit*.
Specie	Organismo de origen del *hit*.
Taxid	Identificador taxonómico del organismo según *NCBI Taxonomy.*
Gene	Nombre del gen asociado a la proteína o anotación del resultado.
Accession	Identificador único de la secuencia en la base de datos (UniProt, Swiss-Prot, RefSeq, etc.). Es el valor utilizado en consultas posteriores.
Range	Rango de posiciones de la secuencia inicio–fin que fue alineada.
Frame	Marco de lectura utilizado para traducir la secuencia de entrada en el análisis BLASTX. Puede tomar los valores +1, +2, +3, –1, –2, –3, los cuales determinan cómo se agrupan los tripletes de nucleótidos al transformarse en aminoácidos y en qué sentido (+/-).
Gaps	Cantidad de posiciones con *"gaps"* en el alineamiento. Son las inserciones o deleciones necesarias para optimizar la alineación entre las secuencias.
Align Len	Longitud total del alineamiento. Indica cuántas posiciones se compararon entre ambas secuencias.

3.2.3.2 Bioconductor

Bioconductor[10] es un proyecto de código abierto basado en el lenguaje de programación R para el análisis de datos en Genómica, como procesamiento de secuencias, análisis estadístico.

Módulo BSgenome

BSgenome[11] es un módulo de Bioconductor que contiene secuencias completas de genomas de referencia. El caso particular de BSgenome.Hsapiens.UCSC.hg38 hace referencia a:

- Hsapiens representa la especie: *Homo sapiens*, ya que el objeto contiene el genoma humano completo en un formato optimizado para la extracción de coordenadas.
- UCSC es la UCSC Genome Browser, el proveedor del ensamblado.
- hg38 *(Human Genome versión 38)* es la versión del genoma humano de referencia.

[10] Bioconductor: https://www.bioconductor.org/
[11] Bioconductor/BSgenome: https://bioconductor.org/packages/release/bioc/html/BSgenome.html

Módulo Biostrings

Biostrings[12] es un módulo de Bioconductor que permite la manipulación de secuencias biológicas (ADN, ARN, proteínas). Por ejemplo:

- *DNAString(), RNAString(), AAString()* permiten crear secuencias.
- *pairwiseAlignment()* para alinear secuencias.
- *matchPattern(), vmatchPattern()* para buscar patrones.
- *alphabetFrequency()* para obtener frecuencias de bases.

Módulo org.Hs.eg.db

org.Hs.eg.db[13] es una base de datos de anotaciones para el genoma humano. Las anotaciones se refieren a metadatos biológicos que describen las características de una secuencia primaria como por ejemplo: la función, localización, variantes, evidencias experimentales, etc. Se puede acceder a este módulo mediante *AnnotationDbi*[14], que permite obtener las claves y columnas disponibles en la base de datos mediante las funciones keys() y columns().

Módulo TxDb.Hsapiens.UCSC.hg38.knownGene

TxDb.Hsapiens.UCSC.hg38.knownGene[15] es una base de datos que contiene anotaciones genómicas del genoma humano orientadas a rangos de secuencias y ubicaciones.

3.2.3.3 Biopython

Biopython[16] es un conjunto de programas informáticos basados en el lenguaje de programación Python con una licencia de código abierto (licencia Biopython) muy poco restrictiva que permite trabajar sobre proyectos relativos al alineamiento de secuencias o cálculo de estructuras proteicas.

Módulo Bio.PDB

Bio.PDB es un módulo diseñado para trabajar con estructuras tridimensionales de macromoléculas (proteínas, ADN, ARN) almacenadas en archivos PDB o mmCIF.

3.2.3.4 AlphaFold

AlphaFold[17] es un sistema desarrollado por *Google DeepMind* que predice con alta precisión la estructura tridimensional de las proteínas a partir de su secuencia de aminoácidos mediante algoritmos de aprendizaje profundo (redes neuronales) para predecir estructuras proteicas con precisión comparable a los métodos experimentales. Además, cuenta con una base de datos con alrededor de 200 millones de predicciones de proteínas por computadora.

---

[12] Bioconductor/Biostrings: https://bioconductor.org/packages/release/bioc/html/Biostrings.html
[13] Bioconductor/org.Hs.eg.db: https://bioconductor.org/packages/release/data/annotation/html/org.Hs.eg.db.html
[14] AnnotationDbi: https://bioconductor.org/packages/release/bioc/html/AnnotationDbi.html
[15] TxDb.Hsapiens.UCSC.hg38.knownGene: https://www.bioconductor.org/packages/release/data/annotation/html/TxDb.Hsapiens.UCSC.hg38.knownGene.html
[16] BioPython: https://biopython.org/
[17] Alphafold: https://alphafold.ebi.ac.uk/faq

RMSD (root-mean-square deviation)

Al comparar dos estructuras proteicas, por ejemplo una predicha con una experimental, se necesita medir qué tan parecidas son. La métrica utilizada por DeepMind es la Raíz de la desviación cuadrática media calculada para el 95% de las mejores distancias.

Para calcularlo se superponen las estructuras y se miden las distancias entre los átomos $C\alpha$ (el esqueleto principal) que son equivalentes y se toma el 95% de las distancias con menor error.

En primer lugar, se alinean las secuencias, se superponen las dos estructuras buscando la rotación/ traslación que permita que ambas estructuras estén superpuestas.
Una vez alineadas, se toma la posición para cada uno de los N átomos $C\alpha$ equivalente en cada modelo, en la posición i:

$$r_i^{\rightarrow exp} = (x_i^{exp}, y_i^{exp}, z_i^{exp})$$
$$r_i^{\rightarrow pred} = (x_i^{pred}, y_i^{pred}, z_i^{pred})$$

La distancia en cada posición i será:

$$d_i = \left|C\alpha_{exp} - C\alpha_{pred}\right| = \sqrt{\left(x_i^{exp} - x_i^{pred}\right)^2 + \left(y_i^{exp} - y_i^{pred}\right)^2 + \left(z_i^{exp} - z_i^{pred}\right)^2}$$

Se toma un subconjunto de las distancias. Para estos se ordenan las distancias entre $C\alpha$ de menor a mayor error y se utiliza solo el 95% con menor error ($N95$), la parte “más confiable”.

$$d_0,\ d_1,\ d_2 \ldots\ d_{N95}$$

Luego se calcula el $RMSD95$ como la raíz cuadrada de: la sumatoria del 95% de las distancias cuadradas con menor error sobre la cantidad de átomos medidos. El resultado está dado en Å (ångström), donde $1\text{Å} = 0.1nm = 10^{-10}\ m$.

$$RMSD95 = \sqrt{\frac{1}{N_{95}} \sum_{i=1}^{N_{95}} d_i^2}$$

Al utilizar carbonos alpha, el $RMSD95$ evalúa cuán bien la predicción reproduce la columna vertebral de la proteína, sin que las regiones muy móviles (las colas desordenadas) distorsionen el promedio. Por esto excluye el 5% peor alineado, ya que se trata de regiones flexibles que incluso en métodos experimentales muestran gran variabilidad.

En el año 2020 se realizó la decimocuarta edición de una competencia internacional para evaluar los métodos de predicción de estructuras de proteínas, la CASP *(Critical Assessment of Protein Structure Prediction)*. En esta edición de la competencia, *AlphaFold2* alcanzó una exactitud comparable a la de métodos experimentales.[18] En esta métrica, las predicciones tuvieron una distancia media de 0.96Å con respecto a los modelos experimentales (siendo 1.4Å el diámetro de un átomo de carbono), en comparación con 2.8Å para el siguiente mejor método.

[18] https://www.nature.com/articles/s41586-021-03819-2

Implementación

Su implementación requiere un entorno de ejecución muy demandante, ya que desplegarlo de forma local implica contar con infraestructura especializada y recursos de hardware muy elevados. Según la página oficial de *AlphaFold* (repositorio de *DeepMind* en GitHub), su instalación es compatible con sistemas Linux y necesita 3 TB de almacenamiento para alojar la base de datos, preferentemente en un disco de estado sólido (SSD) debido al volumen y la velocidad de acceso requeridos. Además, exige una GPU NVIDIA, ya que su capacidad de memoria determina el tamaño máximo de las proteínas que puedan predecirse.

3.2.3.5 Neurosnap

Neurosnap[19] es una web que permite que cualquier persona con una cuenta ejecute *AlphaFold2* en la página o a través de su API utilizando una API key. En la Figura 14 se muestra una imagen de la vista que permite generar y obtener la API Key para utilizar la API de Neurosnap.

El servicio ofrece una versión de prueba gratuita que permite realizar un número limitado de predicciones mensuales utilizando la infraestructura de *AlphaFold*. Esta versión es suficiente para pruebas iniciales, la integración del servicio en un proyecto o la validación del funcionamiento de consultas remotas, aunque presenta ciertas restricciones en cuanto a volumen de predicciones y recursos asignados.

También incluye visualizaciones interactivas en 3D de todas las predicciones y están disponibles para descargar, accesibles desde la vista Overview.

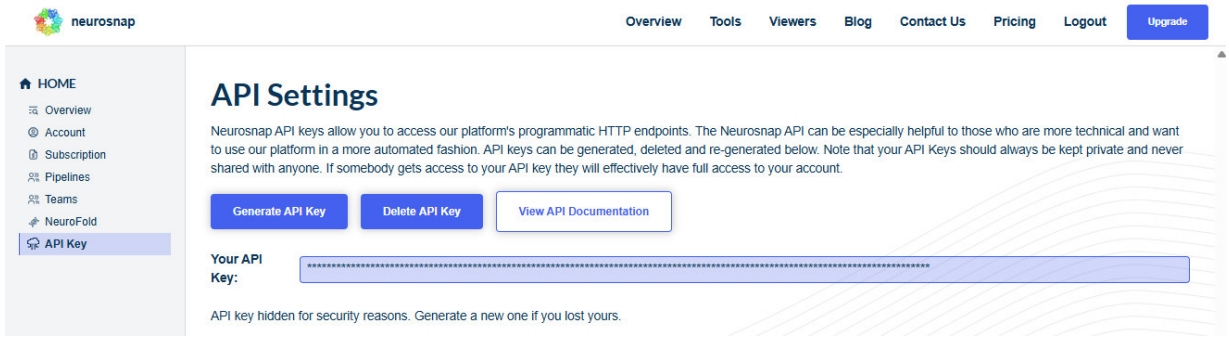

Figura 14. Configuración de la API de Neurosnap.

Nota. Captura de pantalla de Neurosnap (https://neurosnap.ai/overview).

3.2.3.6 3Dmol.js

3Dmol.js[20] es una biblioteca de JavaScript para la visualización molecular en el navegador. Se integra con React y permite mostrar estructuras de proteínas u otras moléculas biológicas. Entre sus características principales se puede mencionar que soporta distintos formatos y

[19] Neurosnap Alphafold2: https://neurosnap.ai/service/AlphaFold2
[20] 3Dmol.js: https://www.npmjs.com/package/3dmol

permite que el usuario interactúe de manera dinámica con la molécula (se pueden hacer rotaciones, zoom, cambiar el estilo de visualización) en tiempo real.

3.2.3.7 Plotly

Plotly[21] es una librería de visualización de datos que permite generar gráficos como diagramas de barras, circulares y *dotplots*, entre otros, para facilitar la interpretación rápida de resultados y mejorar la experiencia del usuario.

3.2.4 Entornos de desarrollo

3.2.4.1 Visual Studio Code

Visual Studio Code[22] (o VS Code) es un editor de código fuente desarrollado por Microsoft y disponible para Windows, Linux, macOS y entornos web. Ofrece soporte para múltiples lenguajes de programación y formatos de archivo como Python, HTML, TypeScript, JSON y YAML. Incluye funcionalidades como resaltado de sintaxis, autocompletado inteligente *(IntelliSense)*, depuración y control de versiones con Git. Además, es posible ampliar sus funcionalidades mediante extensiones, entre las cuales se encuentran linters, herramientas para Docker y complementos.

3.2.4.2 Visual Studio

Microsoft Visual Studio[23] es un entorno de desarrollo integrado (IDE) para Windows y macOS compatible con múltiples lenguajes de programación que incluye compiladores, depuración avanzada y asistentes para la creación de proyectos. Ofrece plantillas específicas para la creación de Web APIs, aplicaciones de escritorio, servicios y otras soluciones basadas en .NET, lo que facilita la configuración inicial y el desarrollo.

3.2.5 Infraestructura y soporte

3.2.5.1 Docker

Docker[24] es una plataforma de contenedorización que permite empaquetar aplicaciones junto a sus dependencias en unidades aisladas llamadas contenedores que garantizan que la aplicación se ejecute en cualquier entorno, independientemente del sistema operativo. Para esto utiliza imágenes, que son plantillas que definen el contenido del contenedor y un motor de ejecución (Docker Engine) que gestiona su creación, ejecución y ciclo de vida. Docker puede utilizarse desde la terminal o mediante herramientas gráficas como Docker Desktop.

---

[21] Plotly: https://plotly.com/javascript/react/
[22] VS Code: https://code.visualstudio.com/
[23] Visual Studio: https://visualstudio.microsoft.com/es/
[24] Docker: https://www.docker.com/

Docker Desktop

Docker Desktop es una aplicación de escritorio que facilita el uso de Docker a través de una interfaz visual para gestionar imágenes, contenedores, logs y configuraciones. Permite inspeccionar contenedores, iniciar pausar o detener servicios y visualizar el entorno de forma más accesible que desde la terminal.

Docker Hub

Docker Hub[25] es un registro público de imágenes Docker que permite publicar y descargar imágenes propias, oficiales y de terceros.

3.2.5.2 Arquitectura cliente-servidor

Esta arquitectura es un patrón de diseño en el que la funcionalidad de una aplicación se divide en dos roles principales que permiten separar las responsabilidades: cliente y servidor. Cuando el cliente realiza una solicitud, el servidor procesa la información, ejecuta la lógica de negocio y devuelve la respuesta al cliente. Por ejemplo, en una aplicación web, el navegador actúa como cliente y el backend RESTful como servidor.

3.2.5.3 Arquitectura en capas

La arquitectura en capas es un estilo de arquitectura de software que organiza el sistema en capas que agrupan responsabilidades relacionadas. Esto mejora la mantenibilidad, facilita el reemplazo de componentes y reduce el acoplamiento. En este modelo, las capas inferiores ofrecen servicios reutilizables mientras que las superiores se concentran en la lógica de negocio y en la interacción con el usuario.

La estructura típica en capas contiene:

- La capa de presentación gestiona la interacción con el cliente externo.
- La capa de aplicación contiene los casos de uso.
- La capa de dominio contiene las entidades e interfaces, es la capa más estable.
- La capa de infraestructura se encarga de la persistencia, la comunicación con las APIs, etc. Materializa las abstracciones definidas en las capas superiores.

El principio fundamental es que las capas superiores dependen de las inferiores, lo cual permite modificar tecnologías de las capas inferiores sin afectar la lógica esencial.

Inyección de dependencias *(Dependency Injection, DI)*

DI es un patrón de diseño que permite que una clase no cree directamente sus dependencias, sino que las reciba desde otro componente, lo que mejora la modularidad y mantenibilidad del sistema reduciendo el acoplamiento entre clases. Por ejemplo, la clase implementa la interfaz que define el contrato y cuando un componente la necesita, se la inyecta.

3.2.5.4 API *(Application Programming Interface)*

Una API es una interfaz que permite que dos sistemas se comuniquen definiendo qué operaciones están disponibles, qué datos se intercambian y qué recibirán a cambio. Expone

[25] Docker Hub: https://hub.docker.com/

funciones para que otros componentes puedan integrarse sin mostrar la lógica interna del sistema, lo cual proporciona encapsulación, interoperabilidad y estabilidad contractual entre distintos módulos de software, servicios o aplicaciones.

3.2.5.5 REST *(Representational State Transfer)*

REST es un estilo arquitectónico para el diseño de servicios web que se basa en recursos a los que se puede acceder mediante URLs, lo que facilita la comunicación entre aplicaciones. Es uno de los estándares más utilizados en APIs modernas, opera sobre los métodos estándar de HTTP como *GET, POST, PUT,* etc. Las respuestas suelen darse en formato JSON.
En conjunto, una API REST aplica los principios REST para exponer las capacidades del sistema de forma predecible y fácilmente consumible.

3.2.5.6 *Middleware* de manejo de excepciones

Un *middleware* es un componente que intercepta las solicitudes HTTP para agregar funcionalidades transversales de manera centralizada, como por ejemplo el manejo de errores.

3.2.5.7 Documentación de APIs

Swagger es una herramienta que permite documentar, probar y consumir APIs REST. Su funcionalidad principal es generar documentación interactiva a partir de la definición de la API y describir endpoints, parámetros y respuestas y esquemas de datos en un formato estructurado. Su interfaz permite probar los endpoints facilitando el desarrollo y la validación de servicios.

3.2.5.8 Fetch API

Fetch es la interfaz estándar del navegador para realizar solicitudes HTTP. Funciona mediante Promesas y permite enviar y recibir datos en formatos como JSON. Es una herramienta central para el consumo de APIs REST dentro de aplicaciones React y otras SPA.

3.2.5.9 HTTP en SPA *(Single Page Application)*

En las aplicaciones de una sola página, cuando el usuario realiza una acción (como ingresar un valor de búsqueda), el frontend envía una solicitud HTTP al servicio. Cuando el servicio responde y el frontend recibe la respuesta, actualiza dinámicamente la interfaz del usuario sin recargar la página completa. Este flujo permite una experiencia interactiva, rápida y eficiente.

3.2.5.10 HTTPS *(HyperText Transfer Protocol Secure)*

El protocolo HTTPS cifra la comunicación entre el cliente (el navegador) y el servidor, lo cual asegura que *requests* y *responses*, incluidos los *headers*, estén cifrados.
En este contexto, *multipart/form-data* se utiliza cuando se requiere envío de archivos, mientras que *application/json* se emplea para datos estructurados como JSON.

3.2.5.11 Navegador

El navegador ejecuta HTML, CSS y JavaScript, y actúa como plataforma de ejecución para aplicaciones web. Incluye motor de renderizado, motor JavaScript y herramientas de

desarrollador. Es el componente responsable de interpretar el frontend, gestionar solicitudes HTTP y mostrar la interfaz al usuario.

3.2.5.12 CORS *(Cross-Origin Resource Sharing)*

Es un mecanismo de seguridad implementado por los navegadores para evitar el Intercambio de Recursos de Origen Cruzado que controla cómo un servidor puede permitir que los recursos sean accedidos desde orígenes distintos al suyo.

Por ejemplo, si un frontend en *http://localhost:5173* quiere consumir una API en *http://localhost:8080,* sin la configuración de CORS el navegador bloquearía la petición. Su configuración permite mantener la seguridad y habilitar la comunicación entre clientes y servidores en distintos dominios.

3.2.5.13 Tecnologías y frameworks

- *React* es una biblioteca de *JavaScript* para construir interfaces de usuario dinámicas basadas en componentes. *TypeScript* le agrega tipado estático mejorando la mantenibilidad y reduciendo errores.
- *Plumber* es un *framework* que permite exponer funciones de R en endpoints HTTP y así facilitar la integración con otros servicios.
- FastAPI es un framework de Python para crear APIs web de manera rápida. Posee soporte para documentación automática y tipado de datos.

3.2.5.14 Controlador de versiones (VCS)

Git es un sistema de control de versiones distribuido que permite registrar los cambios del código fuente a lo largo del tiempo. Con él, cada desarrollador puede trabajar con su copia del repositorio y luego fusionar su trabajo. Permite mantener un historial de modificaciones, crear ramas *(branches)* para desarrollar en paralelo, revertir los cambios y comparar versiones.

Es una herramienta fundamental para mantener la integridad y trazabilidad del código. GitHub es una plataforma que hospeda repositorios Git y permite gestionar las versiones de código, colaborar en proyectos y almacenar repositorios de manera centralizada.

3.3 Contexto histórico y científico del caso de estudio

El raquitismo es una enfermedad causada por la deficiencia de vitamina D, calcio o fósforo[26]. Afecta el desarrollo óseo y puede generar deformidades o fragilidad en los huesos. A partir del descubrimiento de la vitamina D y sus mecanismos metabólicos se comprendió la relación entre la deficiencia de esta vitamina y dicha enfermedad.

El raquitismo tuvo una alta prevalencia en la Patagonia entre las poblaciones no originarias que colonizaron la región a finales del siglo XIX y principios del siglo XX, mientras que se observaron efectos mínimos entre las poblaciones originarias.

---

[26] Roldán, E. (2017). La accidentada historia del descubrimiento de la vitamina D, en su primer centenario. Osteología, 23(1), 49-57.
https://ojs.osteologia.org.ar/ojs33010/index.php/osteologia/article/view/67/49

Es por esto que se plantea el desarrollo de la herramienta para el estudio de esta problemática, para intentar comprender los mecanismos biológicos implicados. Con estos fines, herramientassu bioinformáticas como BLAST, que ayuda a determinar la homología entre secuencias, y *AlphaFold*, que facilita el modelado confiable de estructuras proteicas, resultan de gran utilidad para analizar la variabilidad genética y su impacto en la salud. En este caso particular, la aplicación permite explorar cómo las diferencias en el metabolismo de la vitamina D pueden influir en la salud ósea.

Sección 4: Diseño de solución

4.1 Arquitectura

El diseño de la solución adopta una arquitectura basada en microservicios, favoreciendo la modularidad, escalabilidad y mantenibilidad del proyecto; lo cual permite que cada componente funcione como una unidad independiente con responsabilidades definidas y pueda crecer sin afectar al resto del sistema.

Para garantizar un entorno homogéneo se empleó Docker como tecnología central de contenedorización. Cada microservicio se empaqueta mediante su propio *Dockerfile*, en el cual se declara la imagen base, las dependencias, las librerías y los comandos de inicio. A partir de cada Dockerfile se construye una imagen, que actúa como una plantilla del servicio que, luego, se ejecuta como contenedor, una instancia aislada donde corre el microservicio.

Figura 15. Dockerfile, imagen y contenedor.

La infraestructura se organiza mediante un archivo *docker-compose.yml* que orquesta los cinco contenedores que forman parte de la solución, estos son:

- *web*: Interfaz web desarrollada en React TypeScript y Vite. Consume los endpoints del backend.
- *presentation*: Backend principal desarrollado en .NET. Es el punto de entrada de la solución que centraliza la lógica y comunica el frontend, los servicios internos y realiza consultas a APIs externas.
- *bioc*: Servicio de análisis bioinformático en R con Bioconductor expuesto mediante Plumber para ejecutar funciones estadísticas y análisis de datos genómicos.
- *biopython*: Servicio desarrollado en Python con *FastAPI* y bibliotecas de Biopython para alinear estructuras de proteína y obtener secuencias inversas y complementarias.
- *blast*: Servicio desarrollado con Python y expuesto vía *FastAPI* para la obtención de secuencias, con BLAST, probablemente homólogas a las ingresadas en la aplicación.

A continuación se presenta un gráfico de la arquitectura de contenedores:

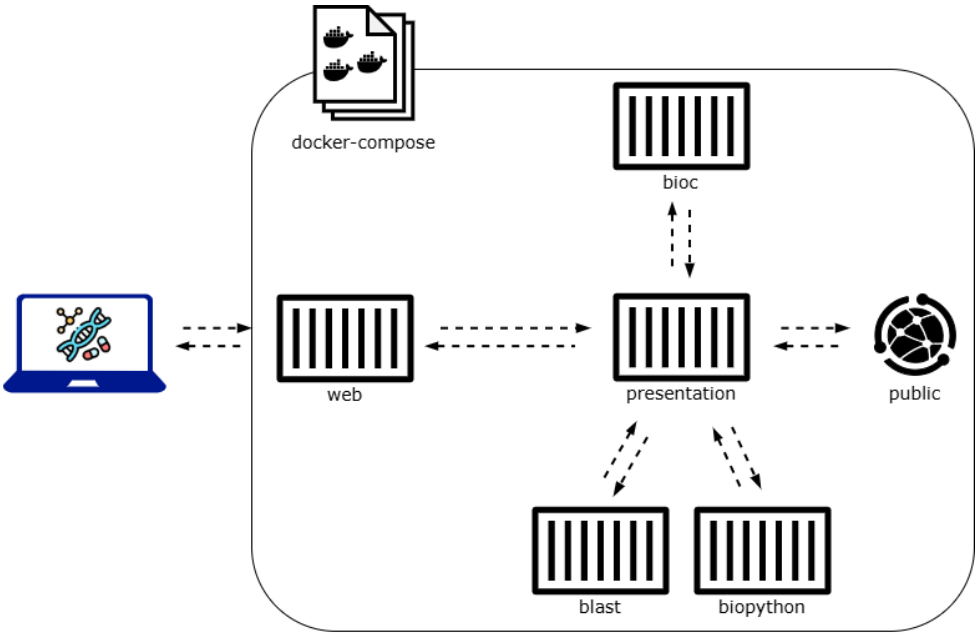

Figura 16. Docker-compose.yml

4.2 Servicio orquestador: .NET

Dentro de la arquitectura, el backend principal se encarga de orquestar la comunicación entre el frontend y los servicios internos y externos. Para ello, se implementa un controlador para cada servicio, una arquitectura en capas que favorece la separación de responsabilidades, el patrón de inyección de dependencias, un middleware para la captura y gestión de excepciones, y objetos de transferencia de datos que estandarizan la comunicación entre el cliente y el servidor. Además, la API se documenta automáticamente mediante *Swagger*.

La Figura 17 contiene el diagrama de clases del servicio orquestador donde se muestra su estructura interna organizada por capas. Incluye controladores, servicios y refleja las dependencias principales entre los componentes. Su propósito es representar cómo se distribuyen las responsabilidades en este servicio.

Figura 17. Diagrama de clases UML

Nota. Elaboración propia (link).

4.3 Servicios internos y externos

La solución integra servicios internos y externos. Los tres servicios internos que trabajan de forma complementaria: *bioc, blast y biopython*. Cada servicio funciona como un microservicio independiente dentro de Docker, expone su propia API y se comunica con el backend principal mediante solicitudes HTTP permitiendo que cada componente evolucione sin afectar a otros módulos. Además, consulta APIs externas para obtener datos genómicos, estructurales y anotaciones de proteínas.

4.4 Comunicación entre microservicios

Para simplificar la comunicación se propone que cada servicio interno devuelva una respuesta uniforme y que, a su vez, el servicio orquestador procese y devuelva una respuesta estándar para mantener un esquema uniforme de respuestas hacia el frontend que, en cada consulta espera cierta estructura definida de antemano.

Sección 5: Planificación

A continuación se describen cronológicamente las tareas planificadas para alcanzar los objetivos propuestos. Cada una representa un hito relevante del desarrollo de la plataforma y está asociada tanto a los resultados esperados, como al tiempo estimado para su finalización.

5.1 Tareas a desarrollar

Definición de requerimientos.

- Relevar las necesidades funcionales y no funcionales del proyecto.
- Identificar fuentes de datos externas.
- Definir el alcance del proyecto.

Diseño de arquitectura.

- Elaborar el diseño general del sistema.
- Definir herramientas.
- Definir los microservicios.
- Resolver la comunicación entre contenedores y los flujos entre el frontend, el backend y los servicios externos.

Definición de módulos frontend y backend.

- Organizar y definir los principales componentes, vistas, controladores, casos de uso y DTOs.

Preparación de Dockerfiles de servicios y docker-compose.

- Construir imágenes para los servicios: .NET, R, Python, BLAST y frontend.

Configuración de contenedores.

- Contenedor .NET: administrar flujos entre servicios y gestión de APIs.
- Contenedor R: ejecutar análisis bioinformáticos en Bioconductor.
- Contenedor Python: implementar funciones de traducción, reverso/complemento y alineamiento estructural.
- Contenedor BLAST: implementar funciones de búsqueda de homología.
- Contenedor frontend (React/TSX): definir la interfaz del usuario, las rutas y adecuar los DTO con los propuestos para el backend.

Exposición de endpoints.

- Definir e implementar las rutas para el acceso a los servicios por parte del servicio orquestador.
- Estructurar modelos de transferencia de datos entre capas y servicios.
- Implementar un flujo simple y completo para verificar la comunicación entre servicios.

Desarrollo del backend .NET.

- Implementar la arquitectura en capas.
- Definir los casos de uso, servicios HTTP client y controladores.
- Agregar el manejo de errores.
- Probar las url, DTOs, autenticación para la conexión con las bases de datos públicas.
- Integrar con la herramienta de predicción de estructura tridimensional (Neurosnap).

Desarrollo del frontend

- Diseñar las vistas principales, rutas, navegación, modales, tablas, loaders, notificaciones, componentes reutilizables.
- Implementar la visualización de gráficos y estructuras.
- Implementar vistas para el ingreso de la API key, el envío de secuencias, el seguimiento del estado de la predicción, la selección de modelos y la recuperación de PDB.
- Implementar servicios de comunicación con la API del backend.
- Integrar con el backend.

Documentación de resultados.

- Describir el funcionamiento del sistema, presentar capturas, diagramas, endpoints y conclusiones.

Caso de estudio: genes asociados a la Vitamina D.

- Ingresar datos reales.
- Analizar secuencias y alineamientos.
- Comparar estructuras de proteínas.
- Analizar resultados.

5.2 Cronograma de trabajo

Tabla 19. Cronograma de trabajo

TAREA	SEMANA															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Definición de requerimientos	X	X	X													
Diseño de arquitectura		X	X	X												
Definición de módulos frontend y backend.				X	X				X	X						
Preparación de Dockerfiles de servicios y docker-compose.			X		X	X					X					
Configuración de contenedores.			X			X	X	X			X					
Exposición de endpoints.			X	X	X				X	X	X					
Desarrollo del backend .NET.		X	X	X	X				X	X			X	X		
Desarrollo del frontend									X	X	X	X	X	X	X	
Documentación de resultados.		X			X				X						X	X
Caso de estudio: genes asociados a la Vitamina D.	X															X

Sección 6: Desarrollo e implementación

El código fuente completo del proyecto se encuentra disponible en el repositorio de GitHub:
GitHub: https://github.com/orgs/vitamina-d/repositories

6.1 Entorno de ejecución y gestión de contenedores

Durante el desarrollo se empleó Docker Desktop como herramienta principal para gestionar los contenedores definidos en el archivo *docker-compose.yml,* lo que permitió visualizar en forma gráfica los contenedores, verificar su estado en tiempo real o reiniciarlos individualmente. En la Figura 18 se muestra una captura de pantalla con los contenedores funcionando.

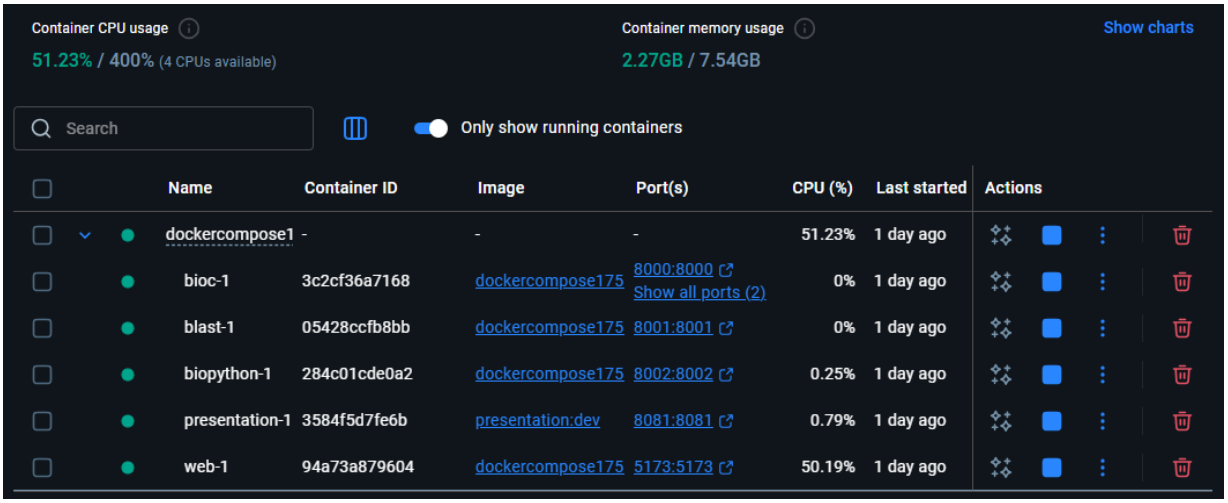

Figura 18. Captura de pantalla de Docker Desktop.[27]

A su vez, para tareas de depuración y pruebas se accedió a los contenedores a través de la terminal con comandos. Lo que permitió inspeccionar la estructura interna de cada servicio, revisar las librerías instaladas, probar su funcionamiento manualmente mediante scripts de prueba y validar configuraciones durante el ciclo de desarrollo. Se accede con el siguiente comando:

```
> docker exec -it <container> bash
```

[27] Docker Desktop: https://www.docker.com/products/docker-desktop/

6.2 Servicios

6.2.1 Servicio bioc

El servicio *bioc* ofrece operaciones vinculadas a la validación de *entrez*, la recuperación de alias, la obtención de características de genes (como tipo, nombre, identificadores o ubicación citogenética), de secuencias y su análisis estadístico o alineamiento.

6.2.1.1 Estructura del repositorio

La implementación del servicio se organiza en un repositorio independiente que contiene el código fuente de la API como muestra la Tabla 20.

Tabla 20. Estructura de archivos bioc

```
bioc/
├── plumber.R
├── endpoints/
│   ├── ...
├── Dockerfile
```

El archivo principal *plumber.R* monta la aplicación y registra los *endpoints* definidos en el directorio *endpoints/,* como muestra la Figura 19:

plumber.R

```r
library(plumber)

#* @apiTitle Vitamina D
#* @apiDescription API para cargar archivos FASTQ, alinearlos con el genoma humano (hg38),

api <- Plumber$new()

api$mount("/autocomplete", Plumber$new("endpoints/autocomplete.R"))
api$mount("/detail", Plumber$new("endpoints/detail.R"))
api$mount("/detailfull", Plumber$new("endpoints/detailfull.R"))
api$mount("/entrez", Plumber$new("endpoints/entrez.R"))
api$mount("/isentrez", Plumber$new("endpoints/isentrez.R"))
api$mount("/stats", Plumber$new("endpoints/stats.R"))
api$mount("/sequence_range", Plumber$new("endpoints/sequence_range.R"))

api$mount("/align", Plumber$new("endpoints/align.R"))
api$mount("/percent", Plumber$new("endpoints/percent.R"))
api$mount("/sequence", Plumber$new("endpoints/sequence.R"))
api$mount("/sequence_and_stats", Plumber$new("endpoints/sequence_and_stats.R"))

api$run(host = "0.0.0.0", port = 8000)
```

Figura 19. Captura de pantalla de plumber.R

6.2.1.2 Imagen Docker del servicio

La ejecución del servicio se realiza mediante una imagen *Docker* personalizada publicada en *Docker Hub*: *veroyols/myapp_bioc:librerias*[28]. Esta, a su vez, se construye a partir de *bioconductor/bioconductor_docker:latest*, la cual incluye *R 4.4.x,* versión compatible con la distribución *Bioconductor 3.20.*

Sobre esta base se instalaron módulos adicionales, entre ellos: *BSgenome, Biostrings, org.Hs.eg.db, AnnotationDbi, TxDb.Hsapiens.UCSC.hg38.knownGene.*

El archivo *Dockerfile* especifica el proceso de construcción, la copia del código fuente al contenedor y la exposición del puerto 8000. La Figura 20 muestra su contenido.

Figura 20. Captura de pantalla del Dockerfile bioc

6.2.1.3 Documentación de APIs

El servicio expone documentación Swagger accesible en http://127.0.0.1:8000/__docs__/ como muestra la Figura 21.

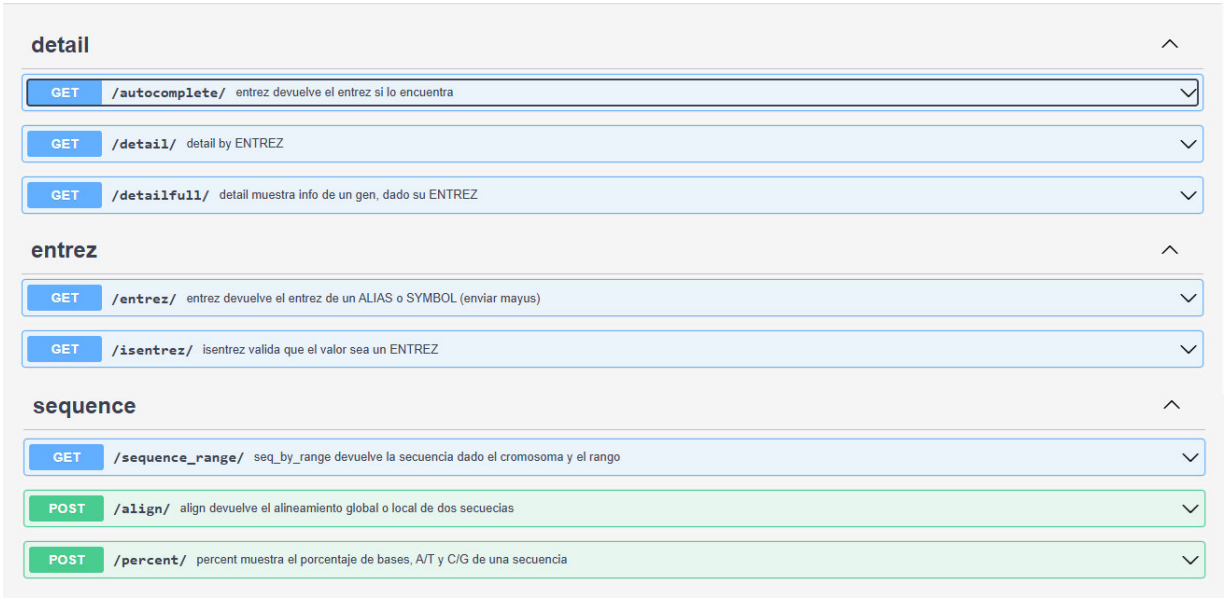

Figura 21. Captura de pantalla de Swagger bioc.

[28] Imagen personalizada `bioc`: https://hub.docker.com/repository/docker/veroyols/myapp_bioc/general

6.2.1.4 Endpoints *bioc*

Cada uno de los endpoints utilizados implementa funciones específicas para validación, recuperación de información y análisis genómico.

/autocomplete

Tabla 21. /autocomplete/

Endpoint	*/autocomplete*
Ruta	*http://bioc:8000/autocomplete*
Método HTTP	GET
Entrada	*?input=string*
Salida	*{code = 200, message = "Ok", data = [aliases...]}*
Códigos de respuesta	*200: Ok* *404: Alias no encontrado* *500: Error de servicio R*
Funcionalidad	Es una función destinada a la sugerencia de alias para facilitar búsquedas predictivas.
Flujo interno	1. Recibe un input. 2. Carga los aliases disponibles para ese valor. 3. Filtra los que coinciden con el texto ingresado. a. Si no hay coincidencias, devuelve 404. b. Si hay, retorna hasta 20 resultados. c. En caso de excepción interna de R, devuelve error 500.

/detail

Tabla 22. /detail/

Endpoint	*/detail*
Ruta	*http://bioc:8000/detail*
Método HTTP	GET
Entrada	*?entrez=int*
Salida	*{* *"code": 200,* *"message": "Ok",* *"data": {* *"entrez": int,* *"symbol": string,* *"genename": string,*

	"genetype": string, *"alias": [string]* *}* *}*
Códigos de respuesta	*200: Ok* *404:* ENTREZ *no encontrado* *500: Error de servicio R*
Funcionalidad	Está orientado a obtener parámetros asociados (como *SYMBOL, GENETYPE, GENENAME, CITOGENETIC LOCATION, ALIAS, UNIPROT IDS*, etc.) a partir del identificador ENTREZ.
Flujo interno	1. Recibe un valor 2. Consulta org.Hs.eg.db usando el ENTREZ como clave. a. Si no hay resultados, devuelve 404. 3. Extrae symbol, genename, genetype y aliases. 4. Retorna un JSON estandarizado con los datos.

/detailfull

Tabla 22. /detailfull/

Endpoint	*/detalfull*
Ruta	*http://bioc:8000/detailfull*
Método HTTP	GET
Entrada	*?entrez=int*
Salida	*{* *"code": 200,* *"message": "Ok",* *"data": {* *"entrez": int,* *"symbol": string,* *"genetype": string,* *"genename": string,* *"citogenetic": string,* *"location": [* *{ "strand": "+|-", "seqnames": "chr", "start": number, "end": number, "length": number }* *],* *"alias": [string],* *"ensembl_id_gene": [string],* *"ensembl_id_protein": [string],* *"uniprot_ids": [string]* *}* *}*

Códigos de respuesta	*200: Ok* *404: Entrez no encontrado* *500: Error de servicio R*
Funcionalidad	Devuelve la información completa de un gen humano a partir de su ENTREZ ID, incluyendo anotaciones, alias, identificadores en otras bases y posiciones genómicas.
Flujo interno	1. Recibe un valor 2. Consulta org.Hs.eg.db usando el ENTREZ como clave. a. Si no hay resultados, devuelve 404. 3. Obtiene alias, ENSEMBL gene, ENSEMBL protein y UniProt. 4. Consulta TxDb para obtener ubicaciones genómicas y arma un listado de regiones. 5. Retorna un JSON estandarizado con todos los datos.

/entrez

Tabla 24. /entrez/

Endpoint	*/entrez*
Ruta	*http://bioc:8000/entrez*
Método HTTP	GET
Entrada	*?value=string*
Salida	*{* *"code": 200,* *"message": "Ok",* *"data": {"entrez"}* *}*
Códigos de respuesta	*200: Ok* *400: Parámetro faltante* *404: ENTREZ no encontrado* *500: Error de servicio R*
Funcionalidad	Devuelve el ENTREZ ID asociado a un ALIAS o SYMBOL.
Flujo interno	1. Valida que el parámetro exista y lo convierte a mayúsculas. 2. Busca el ENTREZ usando *keytype* ALIAS. a. Si lo encuentra, devuelve el *entrez*. b. Si no, intenta con el *keytype* SYMBOL. c. Si no hay coincidencias, devuelve 404. 3. Retorna los ENTREZ ID encontrados en una lista.

/isentrez

Tabla 25. /isentrez/

Endpoint	*/isentrez*
Ruta	*http://bioc:8000/isentrez*
Método HTTP	GET
Entrada	*?value=string*
Salida	*{ "code": 200, "message": "Ok.", "data": { "is_entrez": true | false } }*
Códigos de respuesta	*200: Ok* *500: Error de servicio R*
Funcionalidad	Permite consultar si un valor corresponde con un identificador entrez. Utiliza las anotaciones de org.Hs.eg.db mediante AnnotationDbi.
Flujo interno	1. Obtiene todos los ENTREZID disponibles desde org.Hs.eg.db. 2. Comprueba si el valor recibido está dentro del listado. 3. Devuelve un booleano dentro de una respuesta estandarizada.

/sequence_by_range

Tabla 26. /sequence_by_range/

Endpoint	*/sequence_by_range*
Ruta	*http://bioc:8000/sequence_by_range*
Método HTTP	GET
Entrada	*?chrom=string&start=int&end=int*
Salida	*{ "code": 200, "message": "Ok.", "data": [{ "index": int, "start": int, "end": int, "sequence_length": int, "sequence": "string" }] }*

Códigos de respuesta	*200: Ok* *400: Parámetros inválidos o rango inválido* *404: Cromosoma inexistente* *500: Error de servicio R*
Funcionalidad	Permite obtener una secuencia, dadas sus las coordenadas dentro del genoma.
Flujo interno	1. Valida la presencia del parámetro *chromosome* y convierte *start/end* a enteros. 2. Verifica que el cromosoma exista en BSgenome.Hsapiens.UCSC.hg38. 3. Valida que el rango esté dentro de los límites del cromosoma. 4. Extrae la secuencia entre *start* y *end*. 5. El servicio devuelve una respuesta JSON estandarizada con la secuencia y su longitud.

/align

Tabla 27. /align/

Endpoint	*/align*
Ruta	*http://bioc:8000/align*
Método HTTP	POST
Entrada	*{* *"pattern": "string",* *"subject": "string",* *"type": "global | local | overlap",* *"gapOpening": number,* *"gapExtension": number* *}*
Salida	*{code = 200, message = "Ok.", data = {score, pattern_align, subject_align}}*
Códigos de respuesta	*200: Ok* *400: BadRequest* *500: Error de servicio R*
Funcionalidad	Permite realizar alineamientos entre secuencias de nucleótidos gracias a la función *pairwiseAlignment(),* la cual permite comparar dos secuencias mediante alineamientos globales o locales. Además, admite penalizaciones configurables por apertura *(gapOpening)* y extensión *(gapExtend)*, para facilitar el análisis de similitud biológica.
Flujo interno	1. Recibe el objeto de entrada. 2. Valida secuencias, tipo de alineamiento y parámetros numéricos.

	3. Ejecuta el alineamiento con pairwiseAlignment. 4. Devuelve el puntaje y las secuencias alineadas.

/percent

Tabla 28. /percent/

Endpoint	*/percent*
Ruta	*http://bioc:8000/percent*
Método HTTP	POST
Entrada	*{sequence:string}*
Salida	*{ "code": 200, "message": "Ok.", "data": { "sequence_length": number, "nucleotides": { "A": number, "C": number, "G": number, "T": number }, "cpg_islands": { "count": number, "start": [number] }, "sequence": "string" } }*
Códigos de respuesta	*200: Ok.* 400: *BadRequest* *500: Internal Server Error*
Funcionalidad	Recibe una secuencia de nucleótidos y devuelve porcentajes de nucleótidos e islas CpG.
Flujo interno	1. Valida que la secuencia de entrada exista. 2. Calcular frecuencia de nucleótidos con *alphabetFrequency().* 3. Busca el patrón CpG para obtener posiciones. 4. El servicio devuelve una respuesta JSON estandarizada o un mensaje de error en caso de falla.

6.2.2 Servicio *blast*

El servicio *blast* permite realizar búsquedas de homología mediante BLASTX utilizando bases de datos preinstaladas y devuelve los resultados de las consultas en un formato JSON estandarizado.

6.2.2.1 Estructura del repositorio

El repositorio del servicio blast contiene el código fuente necesario para montar la API, su organización es similar a la del servicio *bioc* y se muestra en la Tabla 29:

Tabla 29. Estructura de archivos blast

```
blast/
├── blast_api.R
├── endpoints/
│   ├── blastx.R
├── Dockerfile
```

El archivo principal *blast_api.R* inicializa la aplicación Plumber y registra el endpoint definido en el directorio *endpoints/* como muestra la Figura 22:

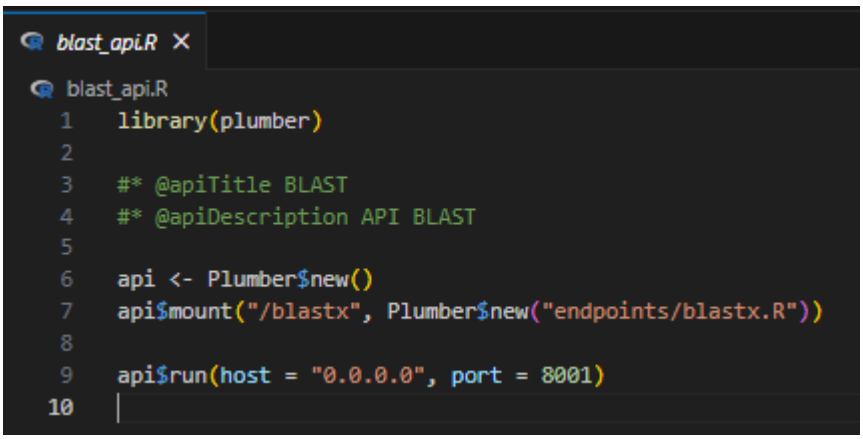

Figura 22. Captura de pantalla de blast_api.R

6.2.2.2 Imagen Docker del servicio

La ejecución del servicio se realiza mediante una imagen *Docker* personalizada construida a partir de *rocker/r-base.* Durante el proceso de construcción se instalaron dependencias como *Plumber*, la distribución local de *ncbi-blast-2.17.0+* y la base de datos *SwissProt*, almacenada en el contenedor en el directorio */opt/blast/swissprot/*. La imagen resultante fue publicada en *Docker Hub: veroyols/myapp_bioc:librerias.*

Para la creacion de la base de datos se siguieron los siguientes pasos:

1. Se accede al contenedor

```
>docker run -it --name blast-container veroyols/blast-r:latest bash
```

2. Una vez dentro, se descarga el archivo FASTA completo de SwissProt, mantenido y distribuido por UniProt. El cual contiene las secuencias proteicas anotadas y curadas manualmente disponibles en UniProtKB.

```
root@8344d41eb30b:/opt/blast/swissprot# wget
https://ftp.uniprot.org/pub/databases/uniprot/current_release/knowled
gebase/complete/uniprot_sprot.fasta.gz
```

```
--2025-11-15 18:11:16--
https://ftp.uniprot.org/pub/databases/uniprot/current_release/knowled
gebase/complete/uniprot_sprot.fasta.gz
Resolving ftp.uniprot.org (ftp.uniprot.org)... 128.175.240.195
Connecting to ftp.uniprot.org
(ftp.uniprot.org)|128.175.240.195|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 93100075 (89M) [application/x-gzip]
Saving to: ‘uniprot_sprot.fasta.gz’

uniprot_sprot.fasta.gz
100%[=================================================>]  88.79M
2.60MB/s    in 35s

2025-11-15 18:11:52 (2.56 MB/s) - ‘uniprot_sprot.fasta.gz’ saved
[93100075/93100075]
```

3. Se verifica su descarga y se extrae:

```
root@8344d41eb30b:/opt/blast/swissprot# gunzip uniprot_sprot.fasta.gz
root@8344d41eb30b:/opt/blast/swissprot# ls
```

```
uniprot_sprot.fasta
```

4. El comando *makeblastdb* permite crear la base de datos de proteínas:

```
root@8344d41eb30b:/opt/blast/swissprot# makeblastdb -in
uniprot_sprot.fasta -dbtype prot -parse_seqids -out swissprot
```

```
Building a new DB, current time: 11/15/2025 18:13:07
New DB name:   /opt/blast/swissprot/swissprot
New DB title:  uniprot_sprot.fasta
```

```
Sequence type: Protein
Keep MBits: T
Maximum file size: 3000000000B
Adding sequences from FASTA; added 573661 sequences in 26.345
seconds.
```

5. Dentro del directorio se genera la base de datos.

```
root@8344d41eb30b:/opt/blast/swissprot# ls
```

```
swissprot.pdb  swissprot.pin  swissprot.pog  swissprot.pot
swissprot.ptf  uniprot_sprot.fasta
swissprot.phr  swissprot.pjs  swissprot.pos  swissprot.psq
swissprot.pto
```

6. Fuera de *blast*, se guardan y suben los cambios del contenedor a Docker Hub.

```
PS C:\Users\veros>     docker commit 8344d41eb30b
veroyols/blast-r:swissprot
```

```
sha256:ab48c75cf8c9e8f31888c79c1906a6953fc5624f4a81b4bda12db83dac716e
5a
```

```
PS C:\Users\veros> docker push veroyols/blast-r:swissprot
```

```
The push refers to repository [docker.io/veroyols/blast-r]
83ac823bb686: Pushing [======>
]  39.53MB/336.3MB
52ce88513a62: Layer already exists
69af0957a6f4: Layer already exists
cb7175931f83: Layer already exists
b2c5e942266c: Layer already exists
1d23be2ddfd0: Layer already exists
93cda266aa27: Layer already exists
5969267de1a1: Layer already exists
```

En la Figura 23 se muestra el Dockerfile correspondiente a este servicio.

Figura 23. Captura de pantalla del Dockerfile blast

6.2.2.3 Documentación de APIs

La API expuesta por el servicio cuenta con documentación generada automáticamente por Plumber, accesible desde la http://127.0.0.1:8001/__docs__/ , como muestra la Figura 24.

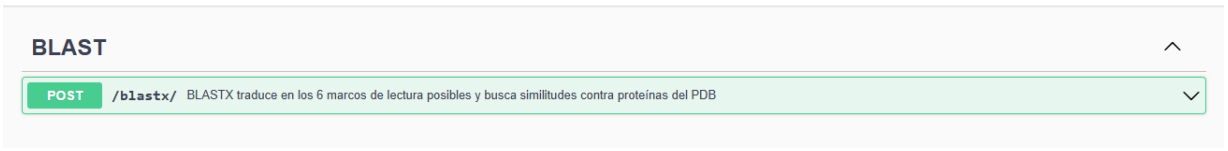

Figura 24. Captura de pantalla de Swagger blast.

6.2.2.4 Endpoints blast

En la Tabla 30 se describe el endpoint */blastx*:

/blastx

Tabla 30. /blastx/

Endpoint	*/blastx/*
Ruta	*http://bioc:8000/blastx*
Método HTTP	POST
Entrada	*{sequence:string}*
Salida	*JSON: {code = 200, message = message, data = result?}*
Códigos de respuesta	*200: Ok.* *500: Error*
Funcionalidad	Contiene la lógica para recibir una secuencia de nucleótidos en formato texto y ejecutar el análisis BLASTX, que consiste en traducir la secuencia en sus seis posibles marcos de lectura para compararla contra la base de datos *SwissProt* (preinstalada en el directorio */opt/blast/swissprot/*) y devolver las mejores coincidencias.

Flujo interno	1. La secuencia se recibe mediante una solicitud POST. 2. El servicio almacena temporalmente la secuencia en memoria. 3. Se construye la consulta especificando: a. el parámetro *"-query"* apuntando al archivo temporal de la secuencia, b. el parámetro *"-db"* con la base de datos de *swissprot* c. y el parámetro *"-outfmt 15"* que solicita la respuesta en formato JSON. 4. La ejecución del comando se realiza dentro de un bloque de manejo de errores *(try–catch)* para capturar fallas . 5. Se limpia el archivo temporal. 6. El servicio devuelve una respuesta JSON estandarizada con el resultado del análisis o un mensaje de error en caso de falla.

La Figura 25 muestra el contenido del archivo *blastx.R*, que implementa esta lógica.

```r
function(sequence) {
    tmpfile <- tempfile(fileext = ".fna")
    writeLines(sequence, tmpfile)
    cmd <- c(
        "-query", tmpfile,
        "-db /opt/blast/swissprot/swissprot",
        "-outfmt", "15"
    )
    out <- tryCatch({
        res <- system2("blastx", args = cmd, stdout = TRUE, stderr = TRUE)
        res
    }, error = function(e) {
        NULL
    })
    unlink(tmpfile)
    if (is.null(out)) {
        response <- list(
            code = 500,
            message = "try catch",
            data = NULL
        )
    } else {
        out_text <- paste(out, collapse = "\n")
        out_text <- iconv(out_text, from = "", to = "UTF-8", sub = "byte")
        result <- tryCatch({
            jsonlite::fromJSON(out_text, simplifyVector = FALSE)
        }, error = function(e) {})
        response <- list(
            code = 200,
            message = "Ok",
            data = result
        )
    }
    return(response)
}
```

Figura 25. Captura de pantalla de blastx.R

En la Figura 26 se presenta la respuesta generada por BLASTX para una secuencia de ejemplo:

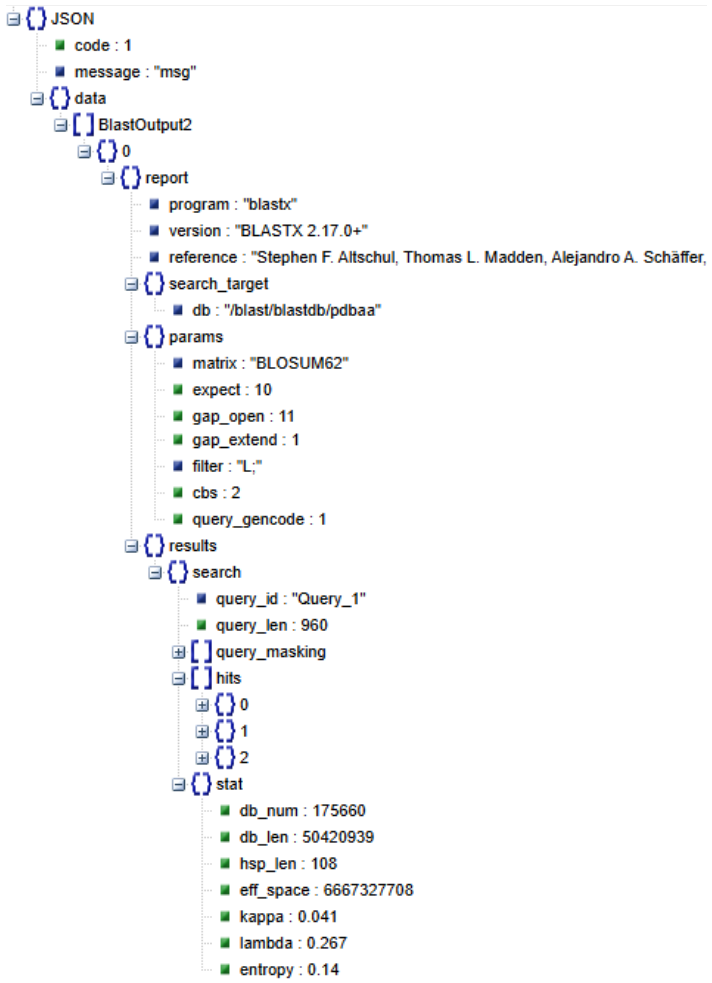

Figura 26. Captura de pantalla del JSON estandarizado de la respuesta blastx (https://jsonviewer.stack.hu/)

6.2.3 Servicio *biopython*

El servicio biopython se expone en el puerto 8002. Su propósito es ofrecer funcionalidades bioinformáticas relacionadas con la traducción y manipulación de secuencias y de análisis estructural de proteínas, utilizando las herramientas del proyecto Biopython.

6.2.3.1 Estructura del repositorio

La estructura del servicio se organiza alrededor del archivo api.py, que inicializa la aplicación *FastAPI* y monta los *endpoints* ubicados en el directorio *endpoints/.* En la Figura 27 se muestra el contenido del archivo *api.py.*

```python
from fastapi import FastAPI
from endpoints import translate, reverse_complement, multipart

description = """

## Endpoints:

* **Translate Sequence** (_recibe una secuencia de nucleotidos y un marco de lectura, y la traduce a una secuencia de nucleotidos_).
* **Reverse Complement** (_recibe una secuencia de nucleotidos y devuelve la paralela o antiparalela_).
* **Align Multipart** (_recibe dos pdb files y devuelve el pdb prediction alineado para poder compararlos_).

"""

app = FastAPI(
    title="biopython api",
    description=description,
    summary="biopython api forma parte del proyecto vitamina-d 🚀",
    version="0.0.1",
    contact={
        "name": "GitHub",
        "url": "https://github.com/orgs/vitamina-d/repositories",
    }
)

#endpoints
app.include_router(translate.router, prefix="/dna", tags=["DNA"])
app.include_router(reverse_complement.router, prefix="/dna", tags=["DNA"])
app.include_router(multipart.router, prefix="/pdb", tags=["PDB"])
```

Figura 27. Captura de pantalla de api.py

La Tabla 31 presenta la estructura completa de archivos.

Tabla 31. Estructura de archivos biopython.

```
/biopython
 ├── api.py
 ├── endpoints/
 │      ├── translate.py
 │      ├── reverse_complement.py
 │      ├── multipart.py
 ├── requirements.txt
 ├── Dockerfile
```

6.2.3.2 Imagen Docker del servicio

El contenedor fue construido a partir de la imagen *python:3.13-slim.* Durante el proceso se instalaron las siguientes bibliotecas:

- *biopython*: es una biblioteca bioinformática que permite manipular secuencias de ADN y proteínas, realizar traducciones, generar reversos y complementos, entre otras funciones.
- *fastapi*: es un *framework* para la creación de APIs, con documentación automática mediante Swagger/OpenAPI.

- *uvicorn*: es el servidor encargado de ejecutar *FastAPI*, escuchar en el puerto configurado y manejar las solicitudes HTTP.
- *python-multipart:* es una biblioteca necesaria para que *FastAPI* procese formularios y archivos.

En la Figura 28 se muestra el *Dockerfile* correspondiente:

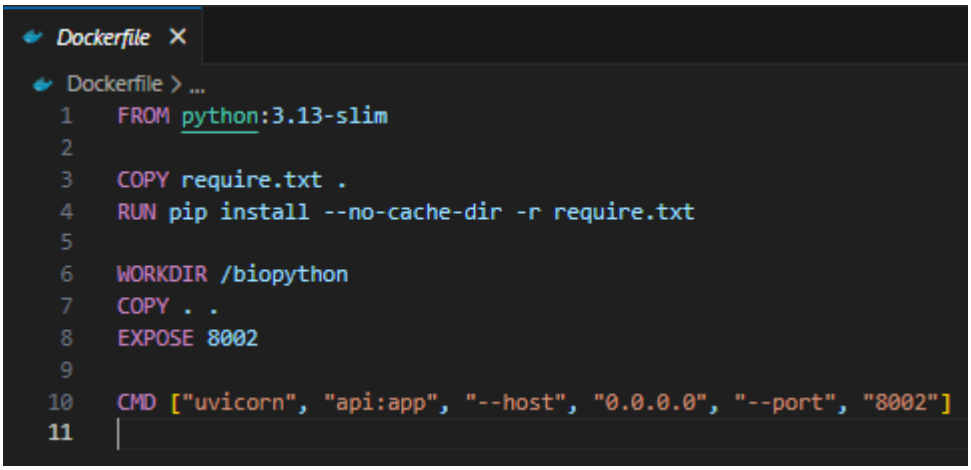

Figura 28. Captura de pantalla del Dockerfile biopython.

6.2.3.3 Documentación de APIs

El servicio expone documentación Swagger accesible en http://127.0.0.1:8002/docs/.

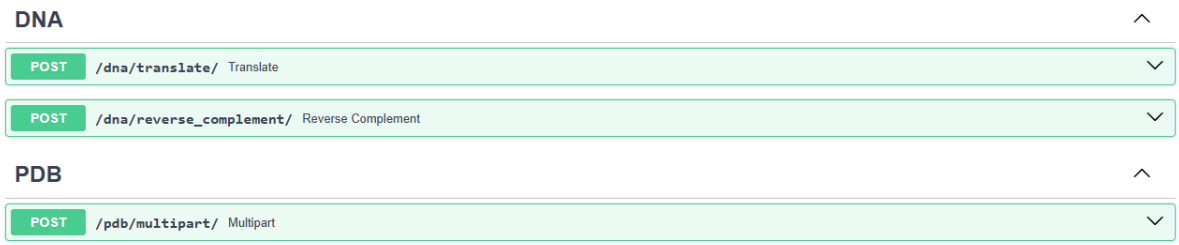

Figura 29. Captura de pantalla de Swagger (biopython).

6.2.3.4 Endpoints biopython

Los endpoints implementados por este servicio son tres y se describen a continuación:

/translate

Tabla 32. /translate/

Endpoint	*/translate*
Ruta	*http://biopython:8002/translate*

Método HTTP	POST
Entrada	*{* *"sequence": "string",* *"frame": 1 | 2 | 3 | -1 | -2 | -3 |* *}*
Salida	*{code = 200, message = message, data = result?}*
Códigos de respuesta	*200: Ok* 400: *TranslationError, ValueError* 504: *TimeoutError* *500: Error*
Funcionalidad	Recibe una secuencia de nucleótidos y un marco de lectura, y devuelve la secuencia de aminoácidos.
Flujo interno	1. Recibe un objeto con la secuencia y el frame. 2. Convierte la secuencia a mayúsculas y valida que los nucleótidos sean permitidos o lanza error. 3. Valida el frame o lanza error. a. Si es positivo, deja la secuencia como esta. b. Si es negativo, la invierte. 4. Traduce la secuencia. 5. El servicio devuelve una respuesta JSON estandarizada con el resultado de la traducción o un mensaje de error en caso de falla.

/reverse-complement

Tabla 33. /reverse-complement/

Endpoint	*/reverse-complement*
Ruta	*http://biopython:8002/reverse-complement*
Método HTTP	POST
Entrada	*{* *"sequence": "string",* *"reverse": true,* *"complement": false* *}*
Salida	*{* *"code": 200,* *"message": "Ok",* *"data": {* *"sequence": "string"* *}* *}*

Códigos de respuesta	*200: Ok* 400: *Bad Request* 504: *TimeoutError* *500: Error*
Funcionalidad	Genera la secuencia invertida, complementaria o reverse-complement según los flags recibidos. Valida que todos los caracteres sean nucleótidos permitidos.
Flujo interno	1. Recibe un objeto con la secuencia y dos booleanos que indican el frame. 2. Convierte la secuencia a mayúsculas. 3. Valida nucleótidos. 4. Aplica reverse, complemento o ambos. 5. Construye respuesta estándar y la devuelve con el resultado de la operación o un mensaje de error en caso de falla.

/multipart

Tabla 34. /multipart/

Endpoint	*/multipart*
Ruta	*http://biopython:8002/multipart*
Método HTTP	POST
Entrada	*{sequence:string}*
Salida	*{code = 200, message = message, data = result?}*
Códigos de respuesta	*200: Ok* 400: *TranslationError, ValueError* 504: *TimeoutError* *500: Error*
Funcionalidad	Alinea dos PDB utilizando sus átomos C-alpha, minimizando el RMSD mediante Bio.PDB.Superimposer y devuelve la prediction alineada
Flujo interno	1. Recibe el archivo PDB de referencia y la predicción como *multipart/form-data.* 2. Lee los bytes y los parsea con PDBParser. 3. Extrae los átomos CA de ambas estructuras. 4. Recorta ambas listas al mismo tamaño. 5. Alinea la predicción sobre la referencia con con *Superimposer.set_atoms()* 6. Genera un archivo PDB resultante en memoria con PDBIO. 7. Devuelve el archivo alineado como *StreamingResponse.*

6.2.4 Servicio *presentation*

Este servicio constituye el backend principal de la solución. Se trata de un proyecto ASP.NET *Core Web API* que actúa como punto de acceso para el frontend y coordina las solicitudes hacia los servicios internos y externos. Incluye controladores REST, *middleware* personalizado, políticas CORS y la posibilidad de iniciar toda la solución en modo depuración mediante Docker Compose. La Figura 30 muestra una captura de la solución organizada en proyectos, uno por cada capa del patrón de capas.

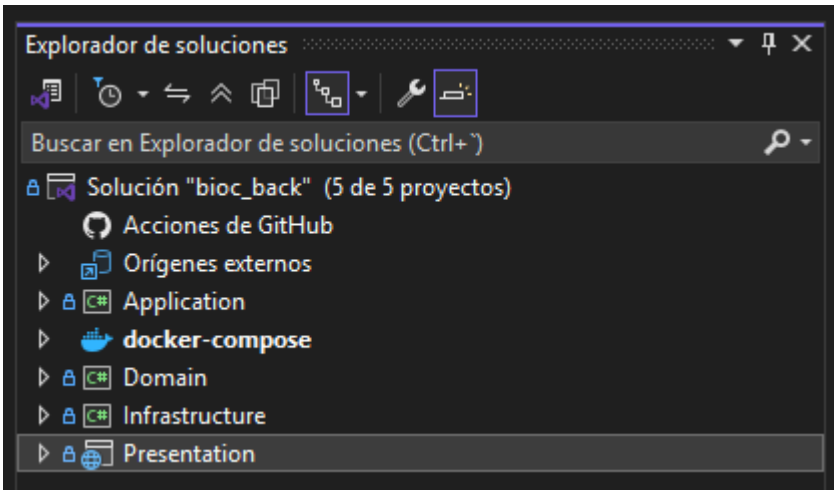

Figura 30. Captura de pantalla de la Solución .NET

6.2.4.1 Patrón de Capas

La aplicación utiliza una arquitectura basada en el Patrón de Capas *(Layers Pattern)* donde la distribución de roles y responsabilidades se organiza de forma jerárquica en cuatro capas principales:

Capa de Presentación

Es la responsable de la interacción con el cliente (en este caso el *frontend*). Sus funciones principales son recibir solicitudes HTTP, validar parámetros básicos, llamar a los servicios de aplicación, estandarizar todas las respuestas *(code, message, data),* manejar errores utilizando un middleware centralizado.

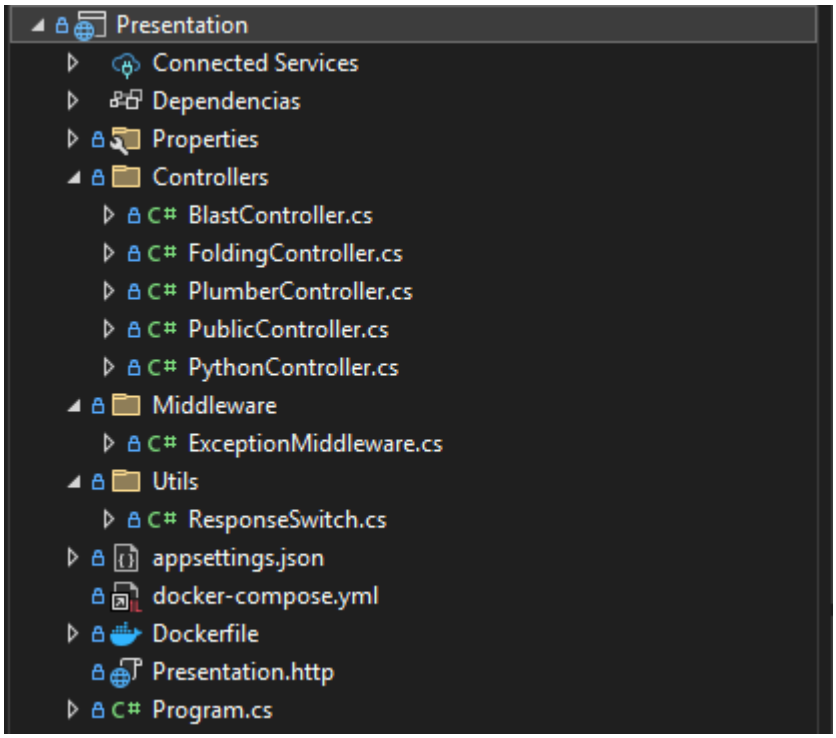

Figura 31. Captura de pantalla de la estructura de carpetas del proyecto presentation

En general, todos los mensajes de entrada y salida se gestionan en formato JSON, excepto los endpoints relacionados con archivos estructurales, que devuelven o reciben archivos *.pdb*.

Capa de Aplicación

Esta capa se corresponde con el núcleo de la aplicación, ya que contiene los casos de uso con la lógica principal. Hace llamadas a la capa de infraestructura (hacia los microservicios internos o APIs externas) y devuelve respuestas normalizadas.

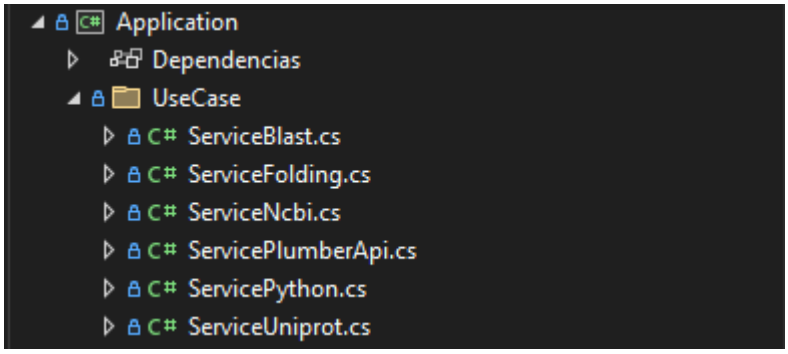

Figura 32. Captura de pantalla de la estructura de carpetas del proyecto application.

Capa de Dominio

Esta capa brinda los elementos que son comunes a toda la solución; como entidades, interfaces, excepciones y objetos de transferencia de datos *(DTO - Data Transfer Objects)* para estandarizar el intercambio de datos y facilitar validaciones.

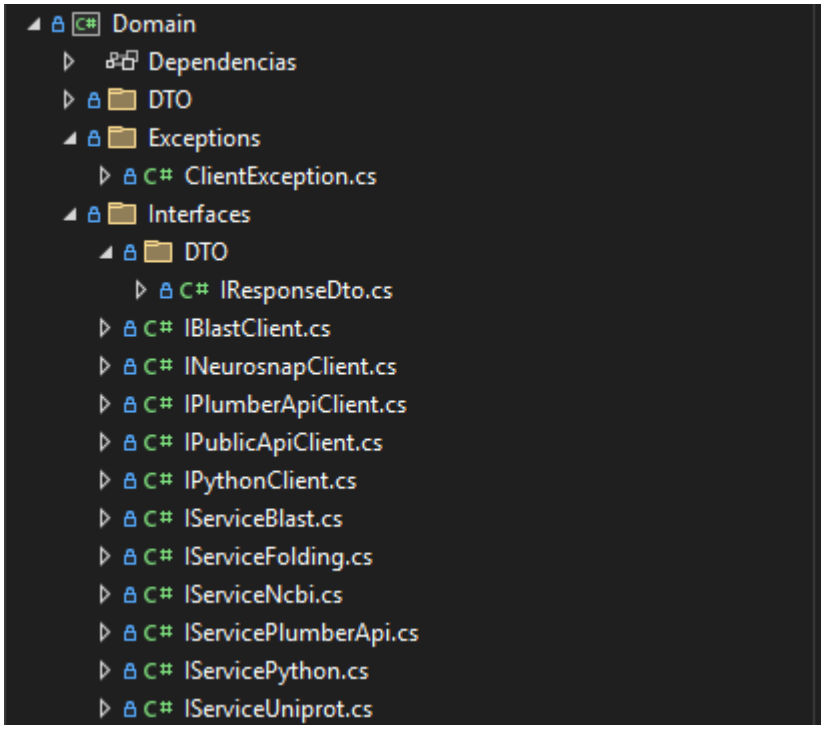

Figura 33. Captura de pantalla de la estructura de carpetas del proyecto domain.

Capa de Infraestructura

Es la encargada de la interacción con sistemas externos. Implementa la configuración de un cliente base y los clientes HTTP hacia microservicios internos *(blastx, bioc, biopython)* y los clientes hacia las APIs externas (UniProt, RCSB, Neurosnap).

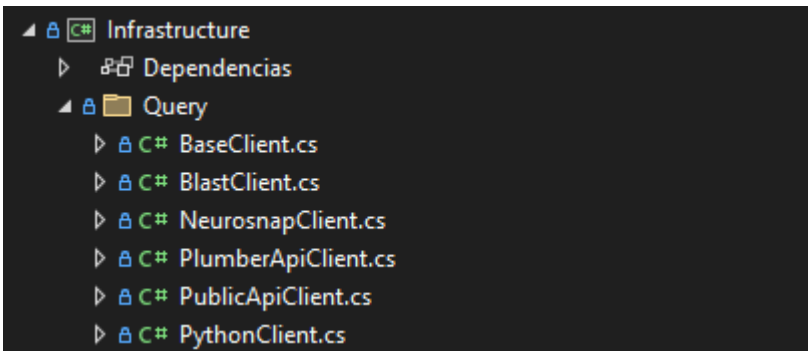

Figura 34. Captura de pantalla de la estructura de carpetas del proyecto infraestructure.

6.2.4.2 Control de CORS

En la aplicación se definió una política de CORS para permitir solicitudes desde los orígenes del frontend, con los *headers* y métodos HTTP habilitados, garantizando que el frontend pueda interactuar con el backend sin restricciones de origen. La Figura 35 muestra la definición de la política en el archivo *Program.cs.*

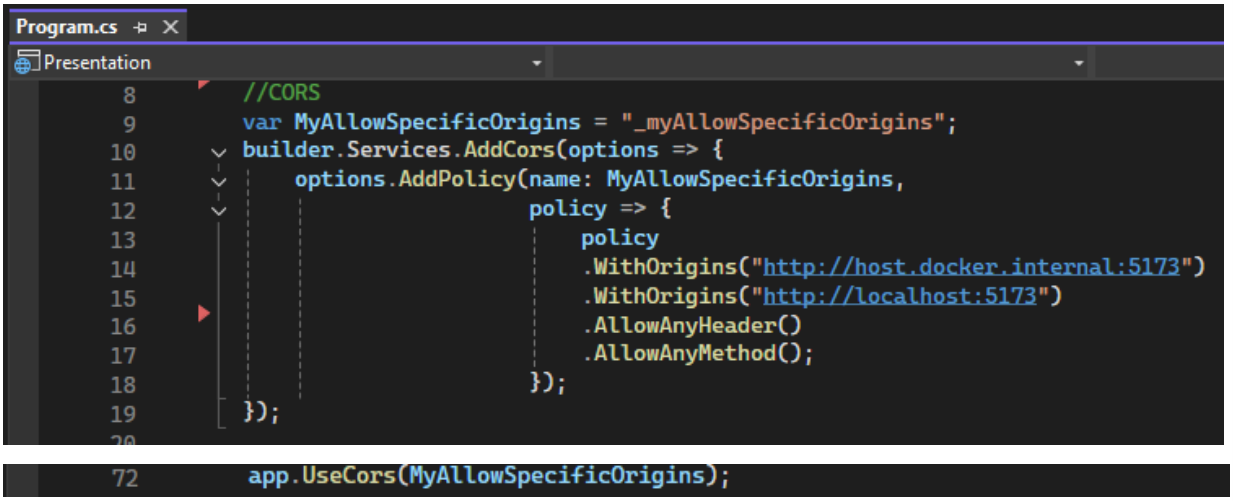

Figura 35. Captura de pantalla de la política de CORS en el archivo Program.cs.

6.2.4.3 Inyección de dependencias

En el archivo `Program.cs` se registran los servicios y clientes HTTP en el contenedor de dependencias para asegurar que todos los controladores y servicios consuman las mismas instancias de manera controlada, evitando dependencias de clases concretas.

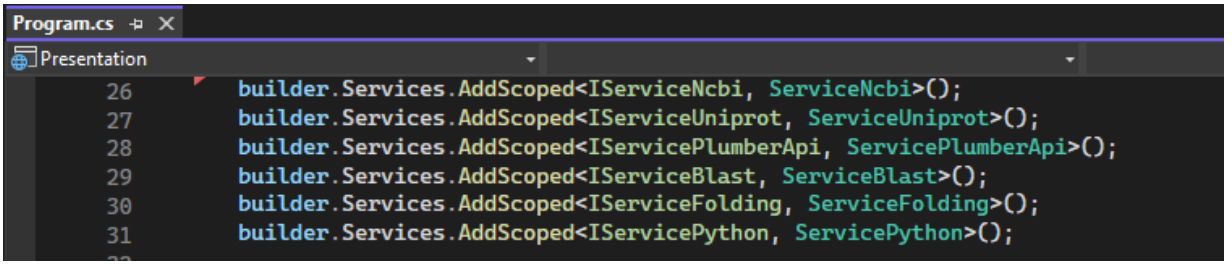

Figura 36. Captura de pantalla del registro de servicios y clientes en el contenedor de dependencias (Parte 1).

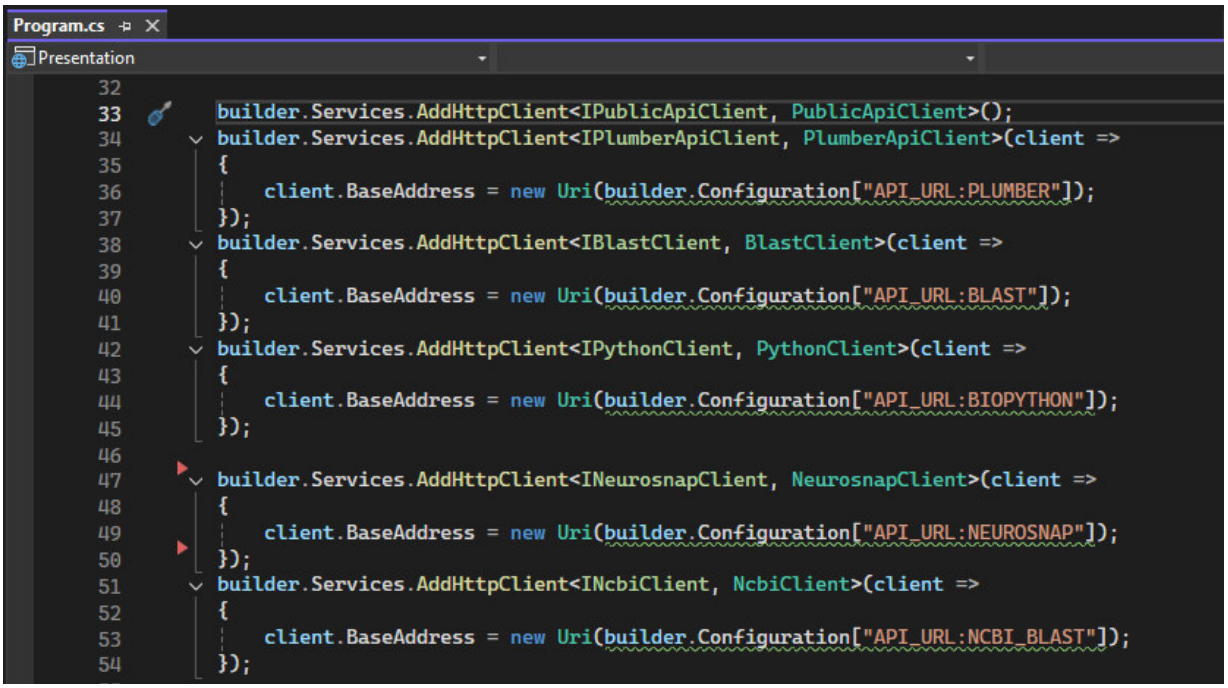

Figura 37. Captura de pantalla del registro de servicios y clientes al contenedor de dependencias (Parte 2).

6.2.4.4 Middleware

Para centralizar el manejo de errores y no duplicar lógica en cada controlador se implementó un middleware de excepciones que captura cualquier excepción no controlada que ocurra durante la ejecución de la solicitud. Lo cual permite devolver respuestas uniformes al cliente, con códigos de estado HTTP apropiados y mensajes estructurados.

Esto facilita el debugging y mejora la mantenibilidad de la aplicación al centralizar la gestión de errores.

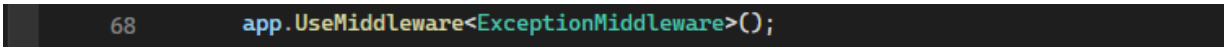

Figura 38. Captura de pantalla de la declaración de Middleware en el archivo Program.cs.

6.2.4.5 Servicios externos

El backend integra diversas APIs utilizadas para obtener datos genómicos, estructurales y anotaciones de proteínas, sin necesidad de mantener bases de datos locales para acceder a datos genómicos, estructuras y anotaciones de proteínas. Estas APIs permiten que la aplicación obtenga información en tiempo real de fuentes confiables sin necesidad de mantener bases de datos locales, lo que garantiza la actualización y consistencia de los datos. Los servicios utilizados son:

NEUROSNAP

URL: https://neurosnap.ai/

Es utilizada por *NeurosnapClient.cs* para iniciar una predicción, verificar su estado, obtener los cinco modelos de la predicción con un valor de incertidumbre, obtener los *pLDDT* para cada residuo de la predicción, descargar el archivo PDB de la predicción.

NCBI E-utilities

URL: https://eutils.ncbi.nlm.nih.gov/
Es utilizada por *PublicApiClient.cs* para obtener los campos necesarios para mostrar el detalle en el frontend. Por ejemplo, la descripcion de un gen.

RCSB PDB

URL: https://files.rcsb.org/
Es utilizada por *PublicApiClient.cs* para descargar el archivo PDB que solicita el *ServicePython.cs* para poder alinear con la predicción de neurosnap.

UniProtKB API

URL: https://rest.uniprot.org/uniprotkb/
Es utilizada por *PublicApiClient.cs* para obtener información cruzada en formato JSON y verificar si un `accession` dado posee una estructura modelada experimentalmente o si se hizo una predicción por computadora. Esto posibilita, dependiendo el caso, ir a buscar el archivo en *RCSB* o en *AlphafoldDB*.

6.2.4.6 Documentación de APIs.

El servicio expone documentación Swagger accesible en *http://localhost:8081/swagger/index.html*.

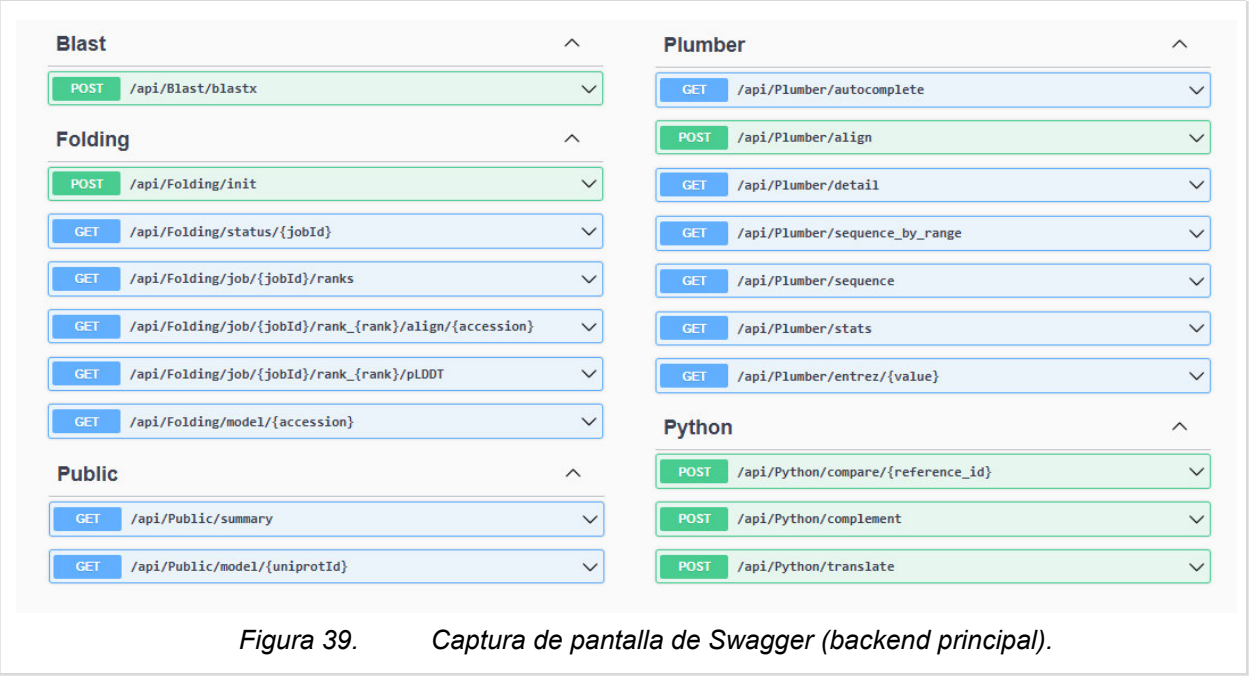

Figura 39. Captura de pantalla de Swagger (backend principal).

6.2.4.7 Endpoints backend principal

Tabla 35. /api/

Ruta	*http://localhost:8181/api/*
Códigos de respuesta	*200: Ok* *400: Bad Request.* *404: Not Found* *500: Internal Server Error* *504: Timeout*

/api/Plumber

Este controlador expone redirige solicitudes al servicio interno bioc, en su mayoría utilizadas por el buscador, la vista de alineamiento, la vista detalle y otras utilidades. Estos son:

Tabla 36. /autocomplete/

Endpoint	*/autocomplete*
Ruta	*http://localhost:8181/api/Plumber/autocomplete*
Método HTTP	GET
Entrada	*Query param:* *input: string*
Salida	*{* *"code": 0,* *"message": "string",* *"data": [* *"string"* *]* *}*

Tabla 37. /align/

Endpoint	*/align*
Ruta	*http://localhost:8181/api/Plumber/align*
Método HTTP	POST
Entrada	*Body:* *{* *"pattern": "string",* *"subject": "string",* *"type": "string",* *"gapOpening": 0,* *"gapExtension": 0* *}*

Salida	*{ "code": 0, "message": "string", "data": { "score": 0, "pattern_align": "string", "subject_align": "string" } }*

Tabla 38. /detail/

Endpoint	*/detail*
Ruta	*http://localhost:8181/api/Plumber/detail*
Método HTTP	GET
Entrada	*Query param: entrez: string in_full: boolean*
Salida	*{ "code": 0, "message": "string", "data": "string" }*
Observación	*Combina detail y detailfull*

Tabla 39. /percent/

Endpoint	*/percent*
Ruta	*http://localhost:8181/api/Plumber/percent*
Método HTTP	POST
Entrada	*Body: "string"*
Salida	

Tabla 40. /sequence/

Endpoint	*/sequence*
Ruta	*http://localhost:8181/api/Plumber/sequence*
Método HTTP	GET

Entrada	*Query param:* *chrom: string* *start: int* *end: int*
Salida	*{* *"code": 0,* *"message": "string",* *"data": {* *"length": 0,* *"nucleotides": { "A": 0, "C": 0, "G": 0, "T": 0, "other": 0 },* *"cpg_islands": { "count": 0, "start": [0] },* *"sequence": "string"* *}* *}*

Tabla 41. /entrez/

Endpoint	*/entrez*
Ruta	*http://localhost:8181/api/Plumber/entrez/{value}*
Método HTTP	GET
Entrada	*Query param:* *value: string*
Salida	*{* *"code": 0,* *"message": "string",* *"data": [* *{"index": 0, "start": 0, "end": 0, "sequence_length": 0, "sequence": "string" }* *]* *}*
Observación	*Combina entrez e isentrez.*

/api/Blast

Este controlador consulta directamente el servicio blast para devolverlo al *frontend.*

Tabla 42. /blastx/

Endpoint	*/blastx*
Ruta	*http://localhost:8181/api/Blast/blastx*
Método HTTP	POST
Entrada	*Body:* *{* *"sequence": "string"*

	}
Salida	*{* *"code": 0,* *"message": "string",* *"data": {* *"entrez": "string"* *}* *}*

/api/Folding

Este controller administra el flujo de predicción de estructuras con *Neurosnap,* inicia la predicción, consulta estados, gestiona descargas de modelos con servicios externos y coordina la alineación posterior con *biopython*.

Tabla 43. /init/

Endpoint	*/init*
Ruta	*http://localhost:8181/api/Folding/init*
Método HTTP	POST
Entrada	*Query Param:* *aminoacid: string*
Salida	*{* *"code": 0,* *"message": "string",* *"data": "string"* *}*
Observación	*Inicia la predicción dada una secuencia de aminoácidos y devuelve el identificador del trabajo iniciado.*

Tabla 44. /status/

Endpoint	*/status*
Ruta	*http://localhost:8181/api/Folding/status/{jobId}*
Método HTTP	GET
Entrada	*Query param:* *jobId: string*
Salida	*{* *"code": 0,* *"message": "string",* *"data": {* *"jobId": "string",*

	"status": "string" } }
Observación	*Devuelve el status de una predicción dado un identificador de trabajo.*

Tabla 45. /ranks/

Endpoint	*/ranks*
Ruta	*http://localhost:8181/api/Folding/ranks/{jobId}*
Método HTTP	GET
Entrada	*Query param: jobId: string*
Salida	*{ "code": 0, "message": "string", "data": { "jobId": "string", "status": "string" } }*
Observación	*Devuelve la lista de predicciones con su valor de incertidumbre.*

Tabla 46. /align/

Endpoint	*/align*
Ruta	*http://localhost:8181/api/Folding/job/{jobId}/rank_{rank}/align/{accession}*
Método HTTP	GET
Entrada	*Route param: jobId: string rank: string accession: string*
Salida	*PDB File*
Observación	*Recupera la URL y la estructura de referencia, la predicción, las alinea y devuelve la predicción alineada con biopython.*

Tabla 47. /pLDDT/

Endpoint	*/pLDDT*

Ruta	*http://localhost:8181/api/Folding/job/{jobId}/rank_{rank}/pLDDT*
Método HTTP	GET
Entrada	*Route param:* *jobId: string* *rank: string*
Salida	*{* *"code": 0,* *"message": "string",* *"data": {* *"plddt": [0],* *"max_pae": 0,* *"pae": [[0]],* *"ptm": 0* *}* *}*
Observación	*Recupera los pLDDT de la predicción de Neurosnap.*

Tabla 48. /model/

Endpoint	*/model*
Ruta	*http://localhost:8181/api/Folding/model/{accession}*
Método HTTP	GET
Entrada	*Route param:* *jobId: string*
Salida	*File PDB*
Observación	*Descarga el pdb de referencia para que el frontend lo muestre con la predicción.*

/api/Public

Expone dos endpoints públicos para la vista detalle del frontend.

Tabla 49. /summary/

Endpoint	*/summary*
Ruta	*http://localhost:8181/api/Public/summary/*
Método HTTP	GET
Entrada	*Query param:* *entrez: string*

Salida	*{ "code": 0, "message": "string", "data": { "name": "string", "description": "string", "summary": "string", "scientificname": "string", "taxId": 0 } }*
Observación	*Descarga el pdb de referencia para que el frontend lo muestre con la predicción.*

Tabla 50. /model/

Endpoint	*/model*
Ruta	*http://localhost:8181/api/Public/model/{uniprotId}*
Método HTTP	GET
Entrada	*Route param: uniprotId: string*
Salida	*File PDB*
Observación	*Descarga el PDB de Uniprot para los uniprotId de la descripción.*

/api/Python

Este controlador consulta los tres endpoints de *biopython* y los devuelve al frontend.

Tabla 51. /complement/

Endpoint	*/complement*
Ruta	*http://localhost:8181/api/Python/complement/*
Método HTTP	POST
Entrada	*Body: { "sequence": "string", "reverse": true, "complement": true }*
Salida	*{ "code": 0,*

	"message": "string", "data": { "sequence": "string" } }
Observación	*Se utiliza para encontrar secuencias complementarias o inversas.*

Tabla 52. /translate/

Endpoint	*/translate*
Ruta	*http://localhost:8181/api/Python/translate/*
Método HTTP	POST
Entrada	*Body: { "sequence": "string", "frame": 0 }*
Salida	*PDB File*
Observación	*Se utiliza antes de predecir la estructura para realizar la traducción de secuencia de nucleótidos a aminoácidos.*

Tabla 53. /compare/

Endpoint	*/compare*
Ruta	*http://localhost:8181/api/Python/compare/{reference_id}*
Método HTTP	POST
Entrada	*Route Param: reference_id: string Multipart Form Data: pdb_file: Form File*
Salida	*PDB File*
Observación	*La funcionalidad que consume este endpoint es utilizada en el flujo de predicción. Sin embargo, fue expuesto para implementar en la vista protein, la comparación visual de dos estructuras ingresadas por el usuario (desde archivo o identificador).*

6.2.5 Servicio web

El servicio web implementa la interfaz de usuario de la plataforma. Es el responsable de gestionar la interacción del usuario con el sistema: la navegación, la presentación de datos, la carga o descarga de archivos FASTA o PDB, el manejo de notificaciones y loaders, y la invocación de los flujos funcionales del análisis genómico.

Consume las APIs expuestas por el backend .NET y muestra información relacionada con genes, estadísticas básicas de secuencias, resultados de los alineamientos y modelos estructurales.

Incluye vistas de búsqueda, detalle, alineamiento, visualización molecular y de análisis estructural.

6.2.5.1 Estructura del repositorio

Como se muestra en la Figura 40, el proyecto bioc-front contiene los archivos de configuración necesarios para el desarrollo y construcción del frontend: package.json, vite.config.ts, .gitignore, el Dockerfile y el archivo HTML base *(index.html).*

Dentro de la carpeta src/ se encuentran los distintos componentes que conforman la aplicación organizados en carpetas por responsabilidades: imagenes, componentes reutilizables, constantes como la paleta de colores, contextos globales de notificaciones o *loaders*, los servicios para consumir las APIs del *backend*, las definiciones de tipos o DTOs, las utilidades, las vistas principales y los wrappers para solicitudes HTTP. Esta estructura favorece la modularidad, mantenibilidad y escalabilidad del proyecto.

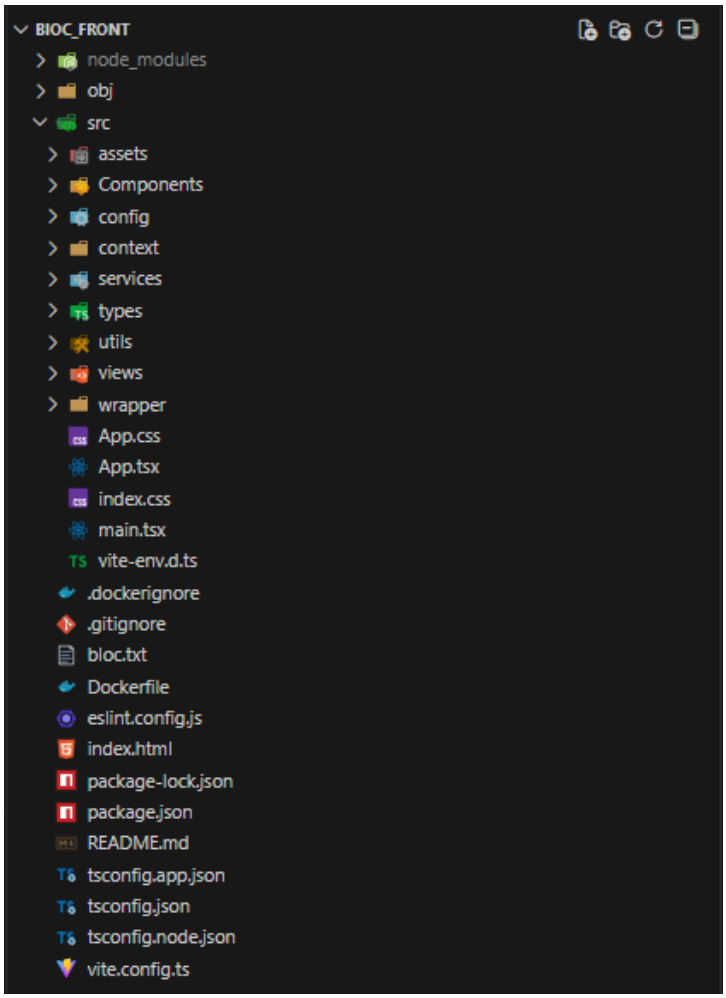

Figura 40. Captura de pantalla de la estructura de archivos web.

6.2.5.2 Imagen Docker del servicio

En la Figura 41 se muestra la captura de pantalla del *Dockerfile web*, que genera el entorno de ejecución del *frontend* utilizando una imagen base de Node y ejecutando el proyecto con Vite.

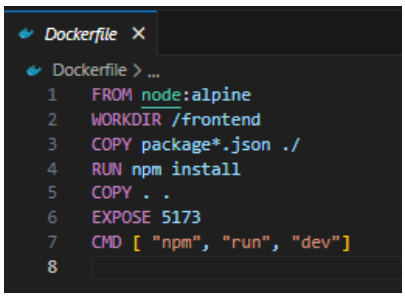

Figura 41. Captura de pantalla del Dockerfile web.

6.2.5.3 Vistas

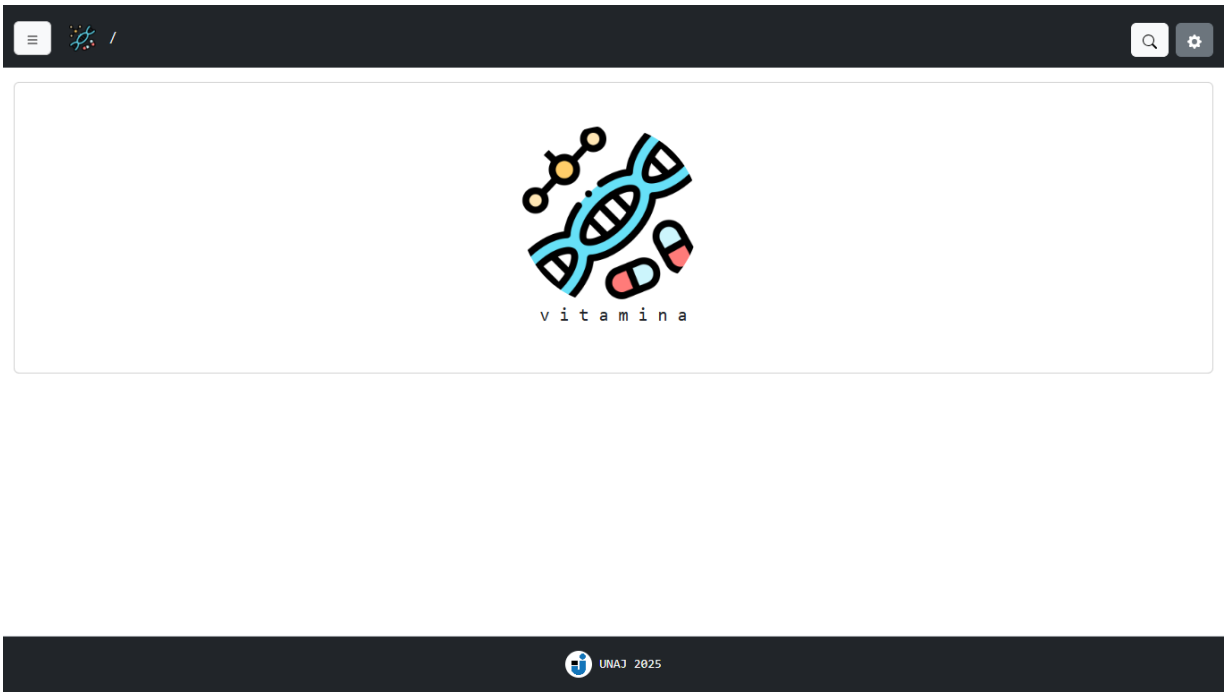

Figura 42. Captura de pantalla de la Vista Home.

Las vistas están organizadas mediante una navegación lateral que contiene accesos directos a la mayoría de las secciones excepto a la vista detalle, a la cual se accede desde el buscador ubicado en la barra superior.

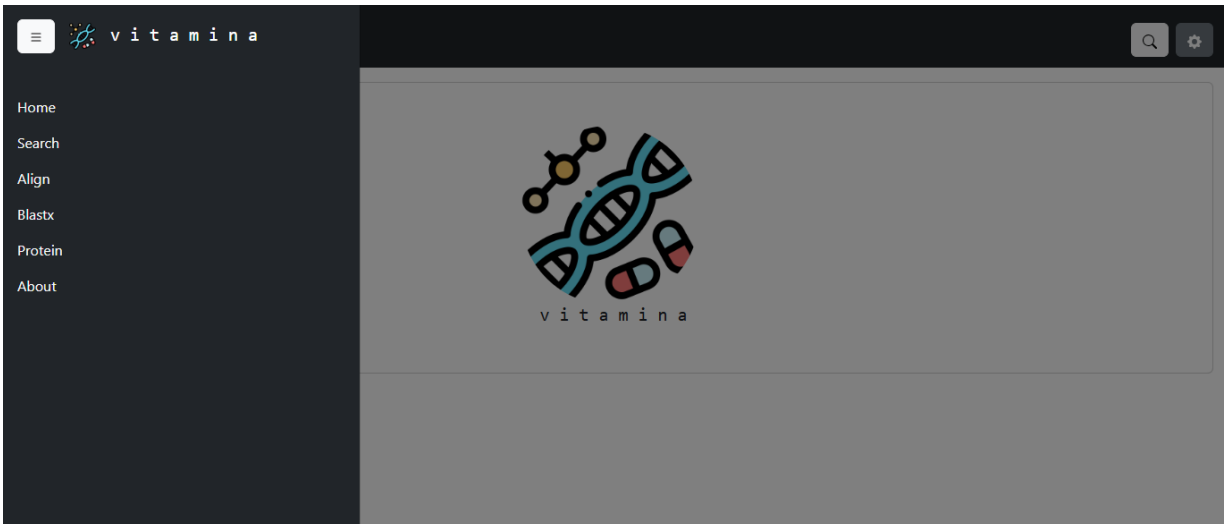

Figura 43. Captura de pantalla de la Navegación.

Vista *blastx*

Esta vista permite ingresar una secuencia de nucleótidos y acompaña todo el flujo hasta la visualización estructural comparativa de: la predicción de la estructura ingresada, alineada con una estructura de referencia de alta similitud, hallada por *blastx* y seleccionada por el usuario.

Los pasos de este proceso son:

1. El usuario ingresa la secuencia manualmente o desde un archivo FASTA.

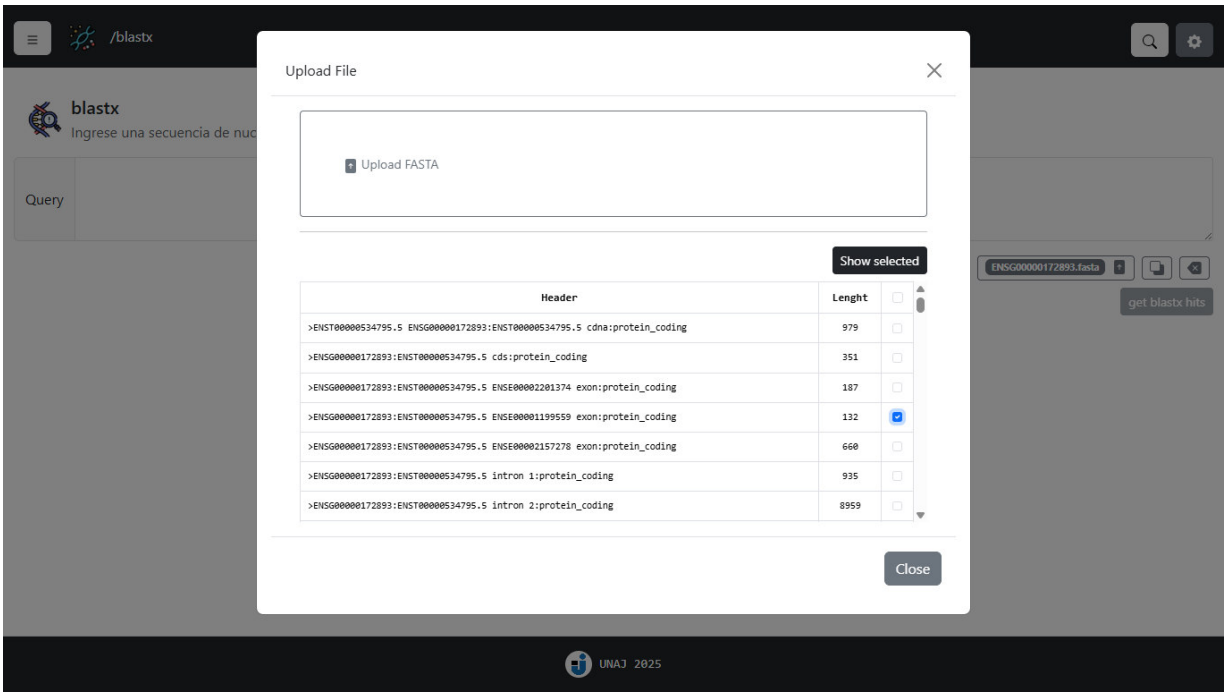

Figura 44. Captura de pantalla del componente Upload Sequence en la vista blastx.

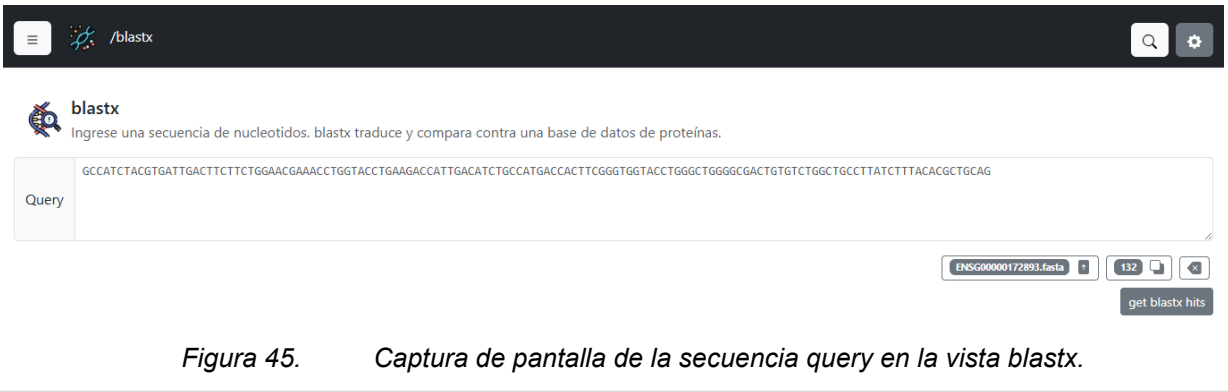

Figura 45. Captura de pantalla de la secuencia query en la vista blastx.

2. La aplicación hace la consulta al backend, a BLASTX vía HTTP REST y presenta los resultados mediante una tabla con las principales métricas y parámetros de cada hit encontrado.

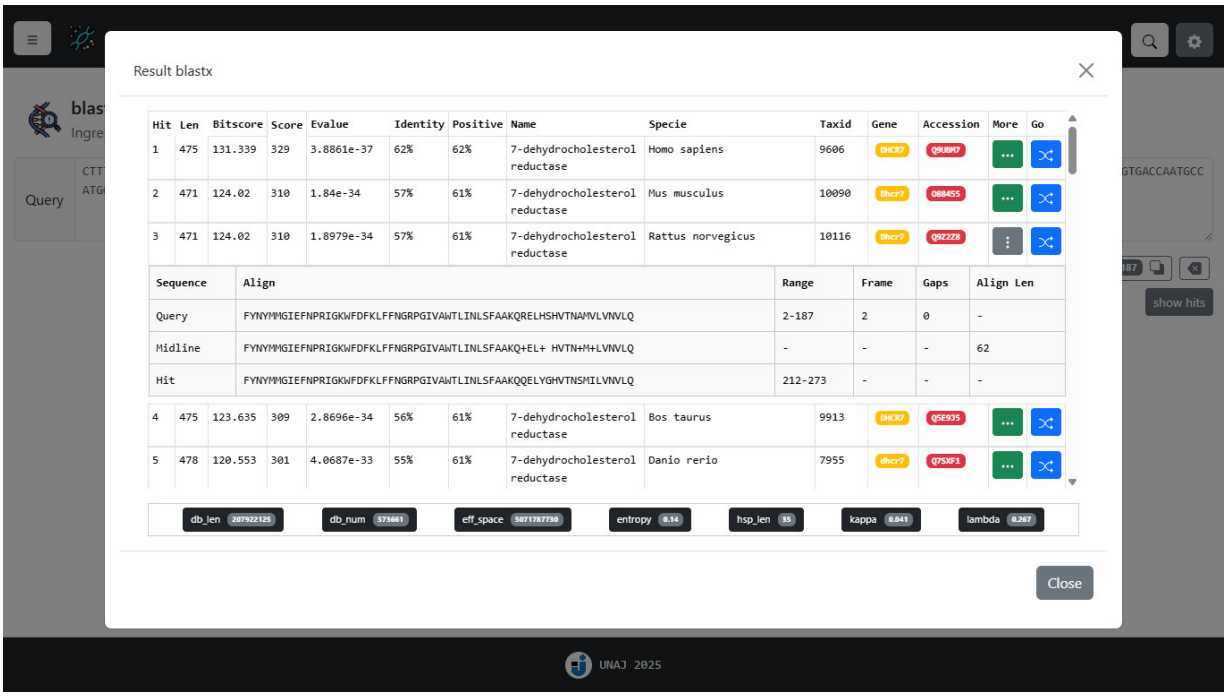

Figura 46. *Captura de pantalla del modal de resultados de blastx en la vista blastx.*

Nota. En la sección 6 se describe cada uno de los parámetros de las columnas de la tabla de hits.

3. El usuario selecciona el *hit* de referencia con el que quiere comparar la secuencia de entrada.
4. El sistema hace una llamada al endpoint de traducción de la secuencia de entrada de acuerdo al *frame* recuperado del *hit* seleccionado.

Figura 47. *Captura de pantalla de la Sección para traducir la secuencia a predecir en la vista blastx.*

5. El usuario puede iniciar el envío de la secuencia de aminoácidos a Neurosnap y visualizar el estado de la predicción (previamente se debe cargar la API key

proporcionada por Neurosnap en la sección *config* que la almacena en la *sessionStorage* para ser enviada en el header al backend).

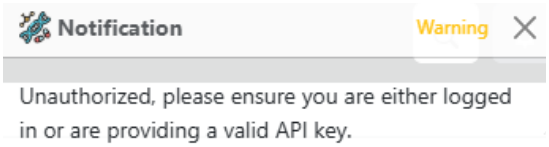

Figura 48. Captura de pantalla de la Notificación para ingresar la API key.

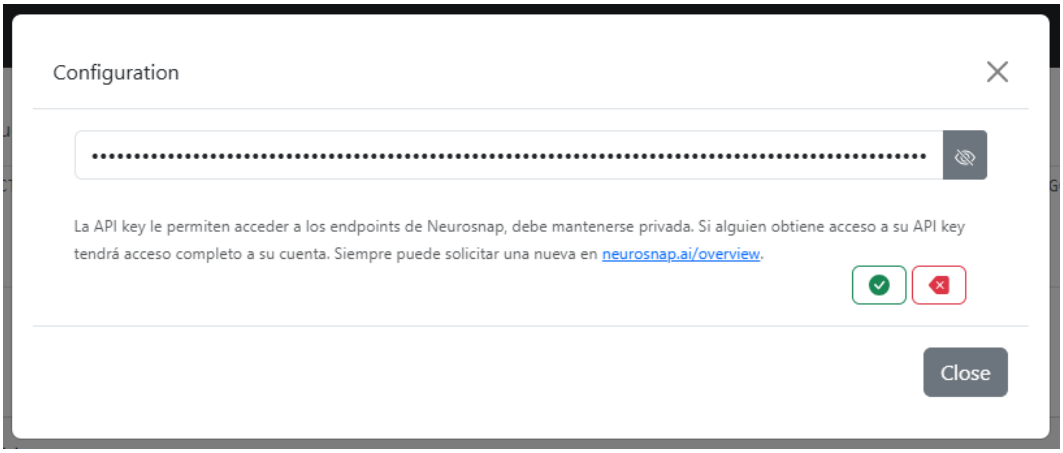

Figura 49. Captura de pantalla del Modal de configuración para ingresar la API Key de Neurosnap.

6. Una vez completada la predicción se puede seleccionar alguno de los cinco modelos generados por Neurosnap con distinto valor de incertidumbre.

Figura 50. Captura de pantalla de la Sección para obtener las predicciones de Neurosnap en la vista blastx.

7. El usuario selecciona una predicción según la incertidumbre y la aplicación hace una solicitud al backend para obtener el PDB predicho por Neurosnap y el PDB de referencia (el hit seleccionado), alineados, y los renderiza en el visualizador *3Dmol.js.* Esta vista permite ver las estructuras secundarias de la secuencia ingresada y la de referencia, visualizar e interactuar con la estructura tridimensional de la predicción y la referencia, descargar el archivo PDB predicho o de referencia y modificar el estilo del gráfico.

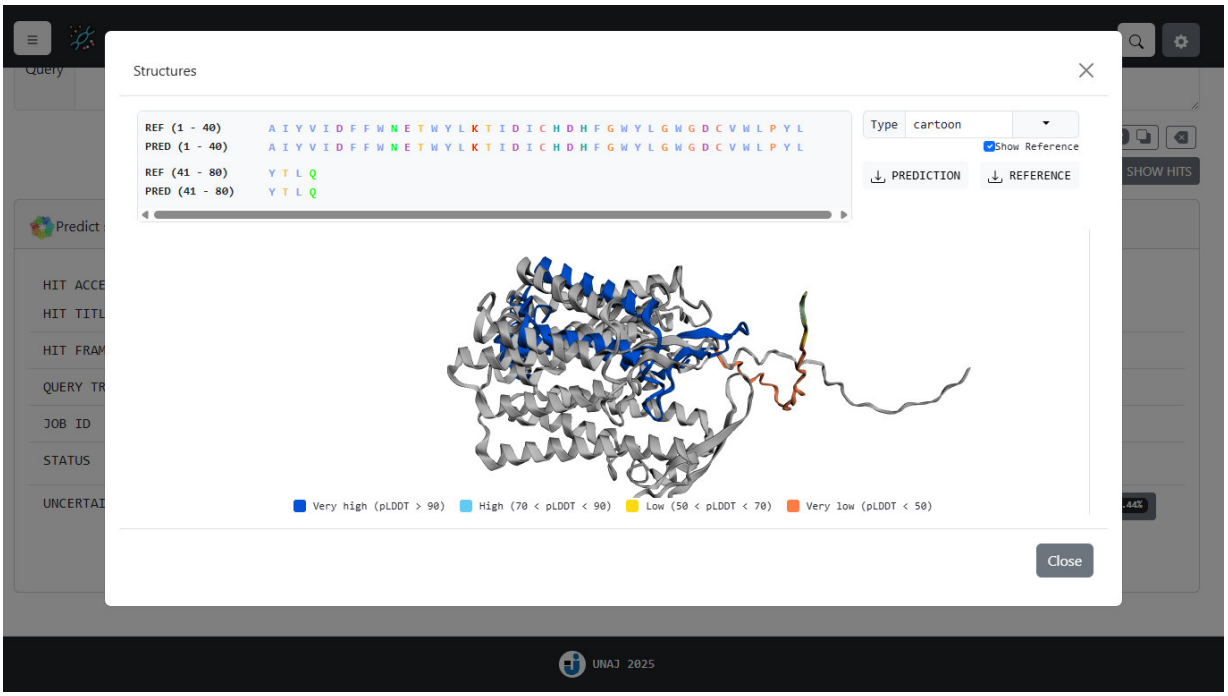

Figura 51. Captura de pantalla del Modal Structure con el Render 3Dmol.js en la vista blastx.

Vista *detail*

La vista de detalle se activa a partir de un modal de búsqueda con cuadro predictivo clickeando en el icono de búsqueda de la barra de navegación superior, en el cual el usuario ingresa un alias de gen. Esta acción permite obtener el ID de Entrez correspondiente y redirigirse a la vista a detalle donde se hace una consulta al backend para mostrar una vista detallada básica que incluye información general del gen como: entrez ID, tipo de gen, nombre del gen, símbolo y alias. Desde esa vista, el usuario puede realizar una nueva consulta para ampliar la información disponible y accediendo a campos adicionales como: descripción que contiene un resumen funcional del gen, ubicación citogenética y los IDs de Uniprot si es que los hay.

Además, se habilita la navegación hacia datos más específicos que incluyen la secuencia completa del gen, las estadísticas de composición de nucleótidos y la visualización de las estructuras proteicas asociadas a cada UniProt ID.

Este flujo permite al usuario explorar progresivamente desde información básica hasta datos estructurales y estadísticos que integran consultas a APIs externas y ofrecen una experiencia interactiva dentro de la aplicación.

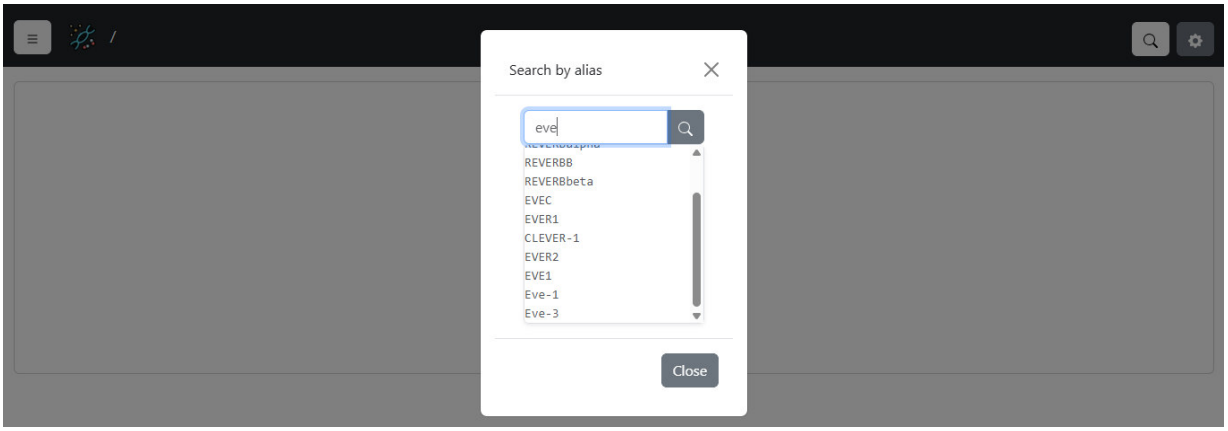

Figura 52. Captura de pantalla del Modal de búsqueda.

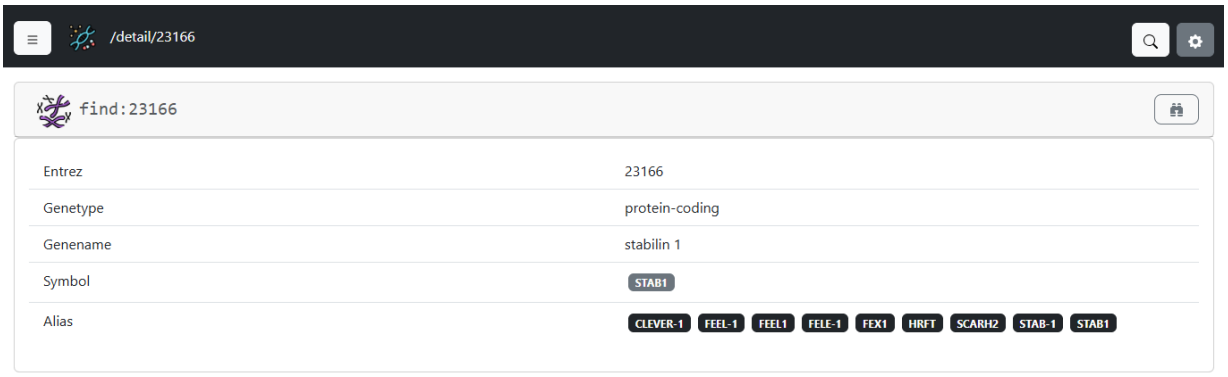

Figura 53. Captura de pantalla del detalle en la vista Detail.

Figura 54. Captura de pantalla del detalle completo en la vista Detail.

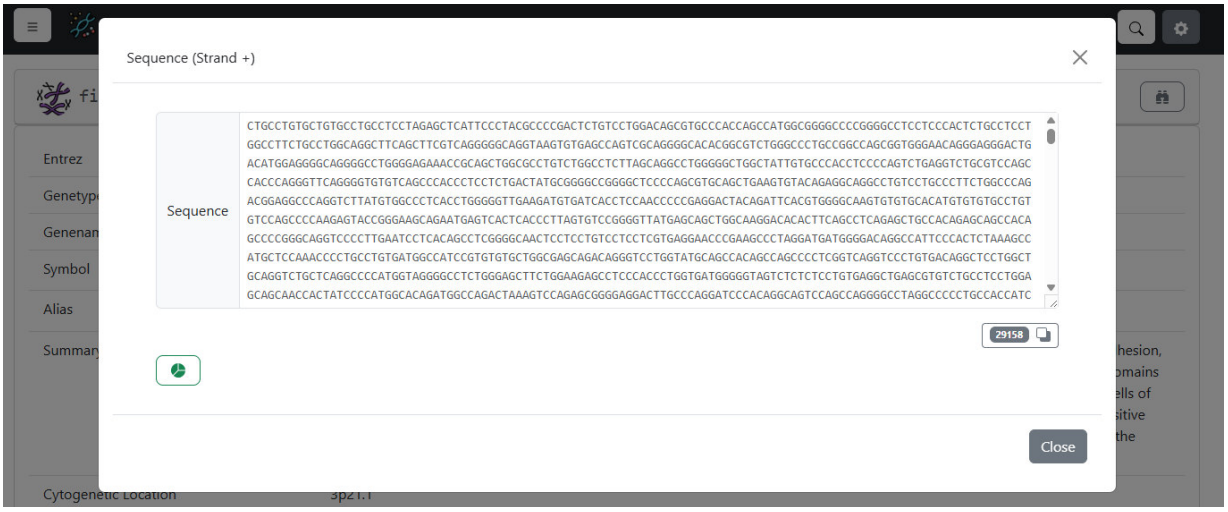

Figura 55. Captura de pantalla de la secuencia del gen en Detail Full.

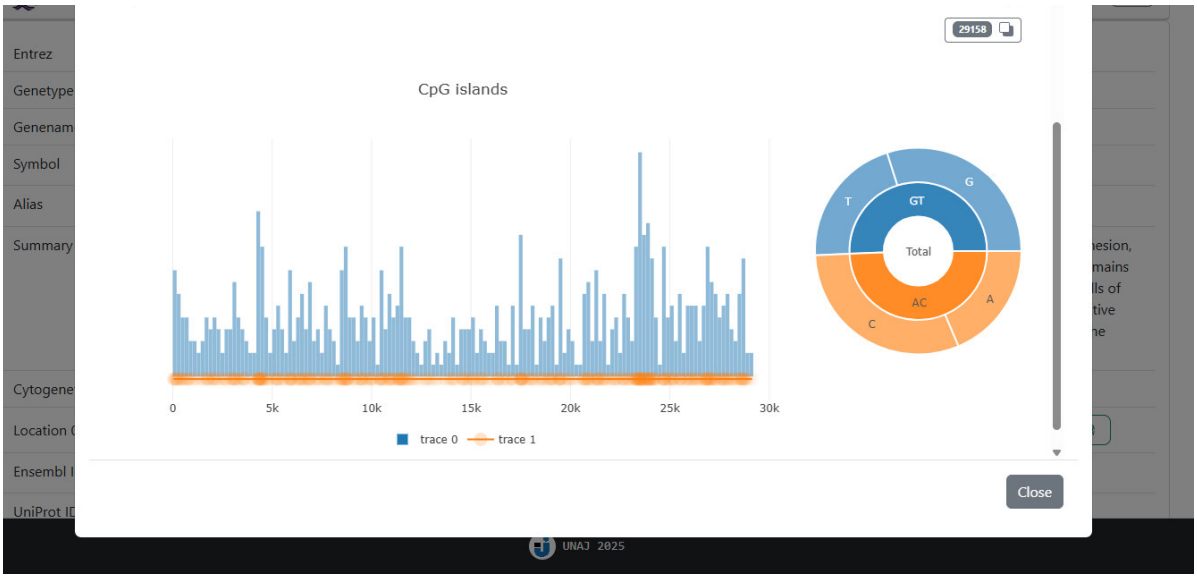

Figura 56. Captura de pantalla de las estadísticas de la secuencia en Detail Full.

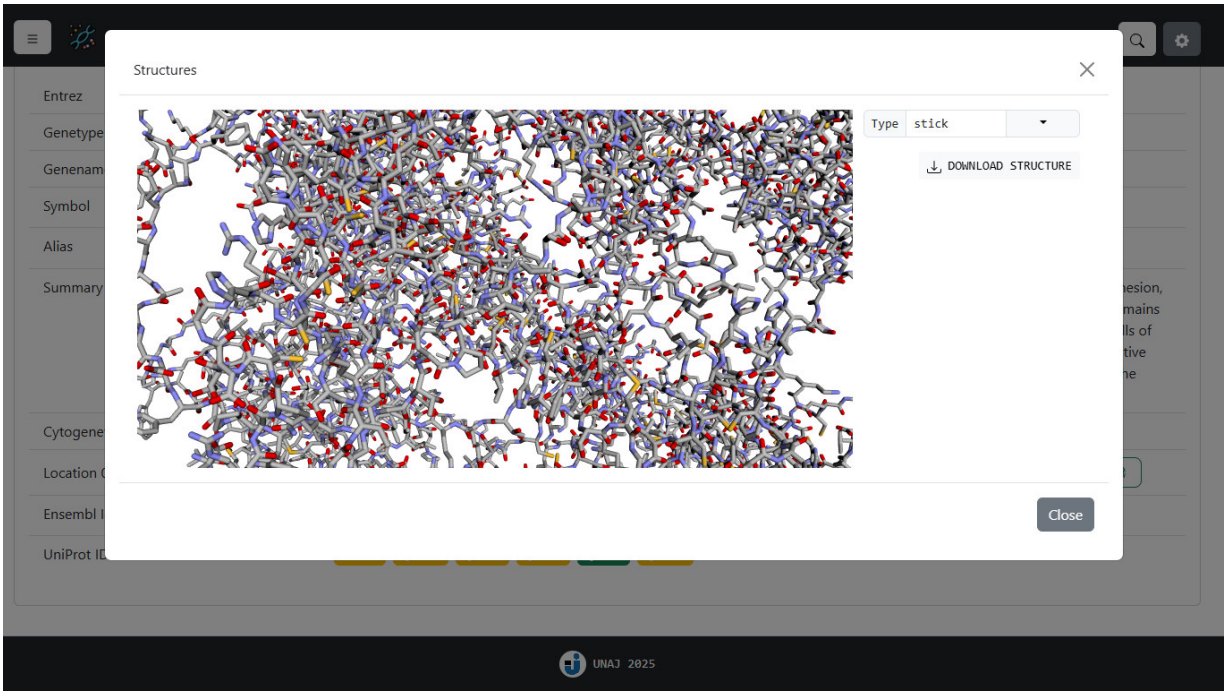

Figura 57. Captura de pantalla del Modal de estructura según el UniprotID en la vista Detail Full.

Vista *search*

La vista Search permite al usuario obtener secuencias genómicas a partir de su ubicación molecular mediante un selector de cromosomas y dos campos de entrada para ingresar el rango deseado, como se muestra en la Figura 58.

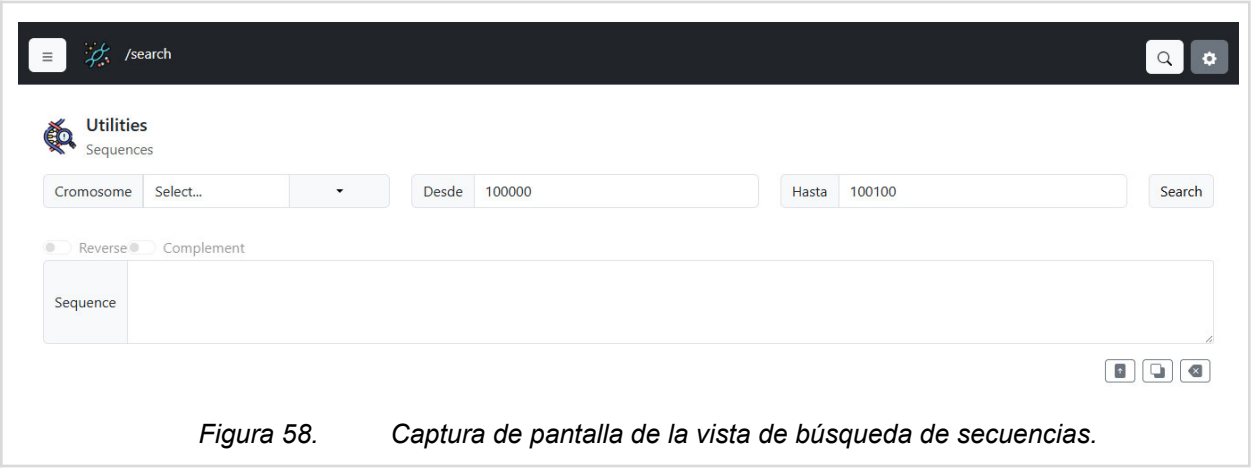

Figura 58. Captura de pantalla de la vista de búsqueda de secuencias.

Adicionalmente, como muestra la Figura 59, el componente que muestra la secuencia ofrece funcionalidades como un *switch* para mostrar la secuencia reversa y/o complementaria en un segundo cuadro, botones para subir una secuencia desde el explorador de archivos o mediante drag and drop, copiar el contenido del cuadro de secuencia o borrar la secuencia cargada.

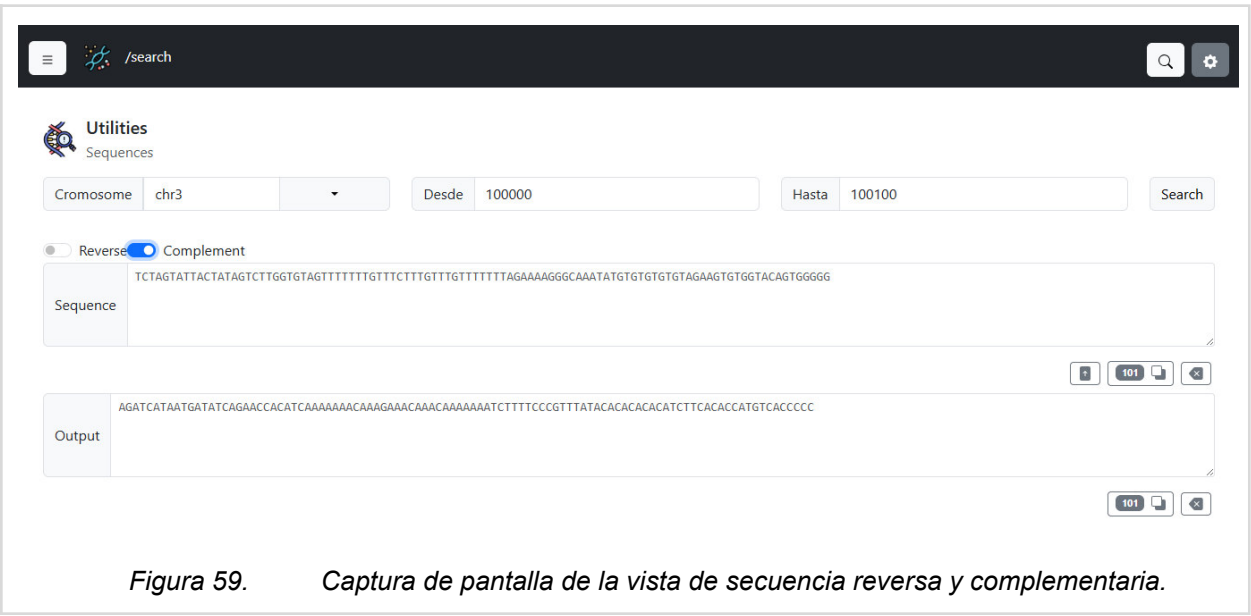

Figura 59. Captura de pantalla de la vista de secuencia reversa y complementaria.

Al subir un archivo, el sistema analiza su contenido y genera una tabla con todas las posibles secuencias presentes, permitiendo al usuario seleccionar la que desea procesar para

facilitar la exploración, manipulación y análisis de secuencias de manera interactiva en la aplicación.

Vista *align*

La vista Align permite al usuario ingresar dos secuencias en cuadros de texto separados y visualizarlas inicialmente en un *dotplot*, mostrando visualmente las posiciones coincidentes entre ambas. Al presionar el botón Alinear, el backend realiza el alineamiento y devuelve las secuencias alineadas. Los resultados se muestran en un plot coloreado por letra, facilitando la visualización de coincidencias y diferencias entre las secuencias de manera interactiva y clara.

Figura 60. Captura de pantalla de la Vista Align.

6.3 Funcionalidades principales

La solución integra los servicios definidos para ofrecer sus funcionalidades principales. A continuación se describen los flujos completos de análisis estructural y de detalle de secuencias, acompañados de los diagramas de procesos que muestran cómo el backend .NET orquesta dichos servicios.

6.3.1 Análisis estructural

6.3.1.1 Diagrama de flujo

El análisis estructural (se corresponde la vista blastx) abarca desde la carga de la secuencia hasta la visualización con una proteína de referencia. El backend .NET coordina la búsqueda de homologías con BLAST, la transformación y alineamiento estructural con Biopython, y la

predicción de estructuras tridimensionales con Neurosnap. La Figura 61 muestra el diagrama de flujo de esta funcionalidad, principalmente desde la perspectiva del frontend.

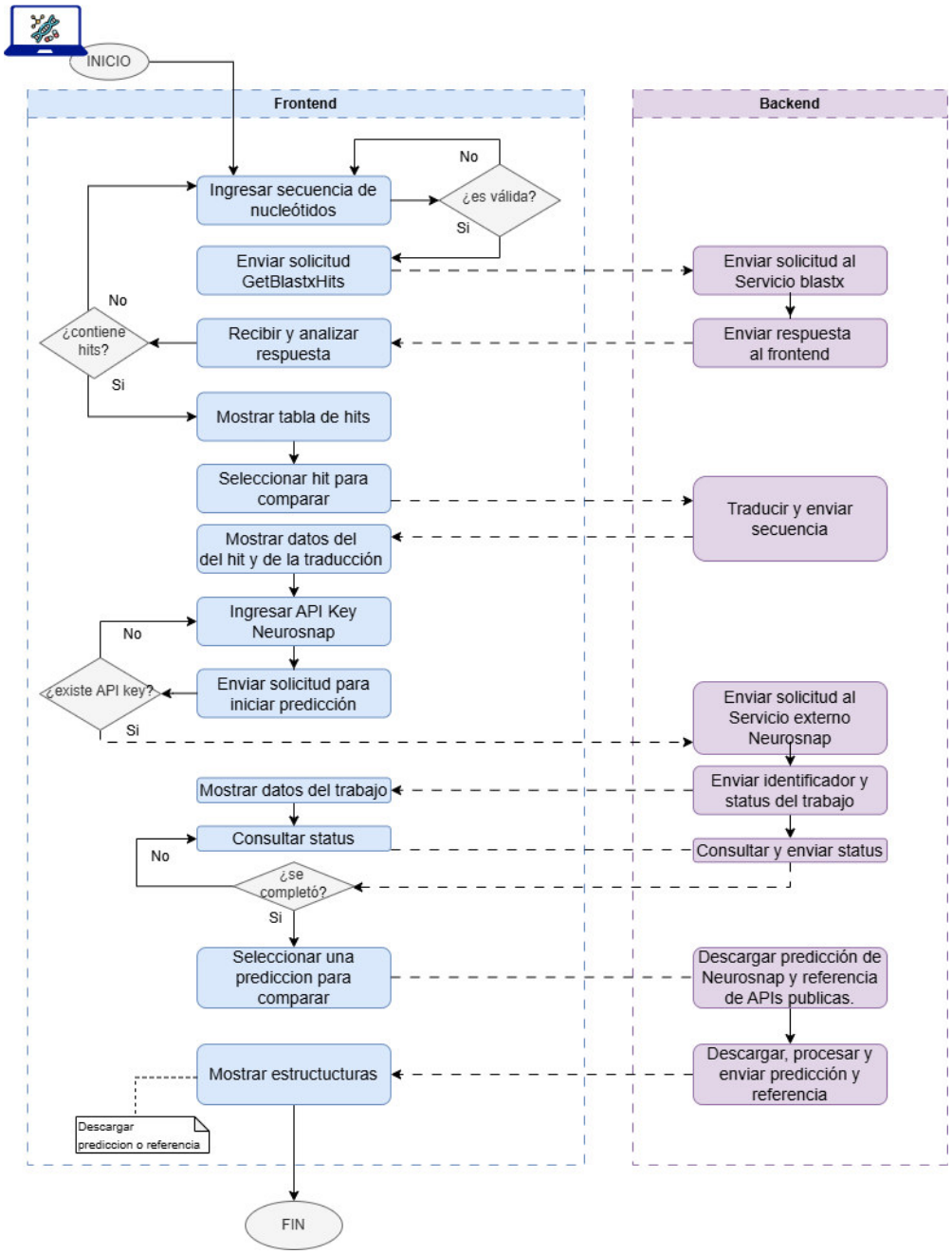

Figura 61. Diagrama de flujo de la funcionalidad de Análisis estructural.

Nota. Elaboración propia (link).

6.3.1.2 Diagrama de procesos

El diagrama de procesos muestra cómo el backend .NET orquesta los microservicios internos y las APIs externas, que interactúan de manera coordinada para procesar la secuencia ingresada por el usuario y visualizar los resultados.

Procesar secuencia con blastx

Este primer endpoint obtiene las secuencias con mayor similitud a la secuencia ingresada.

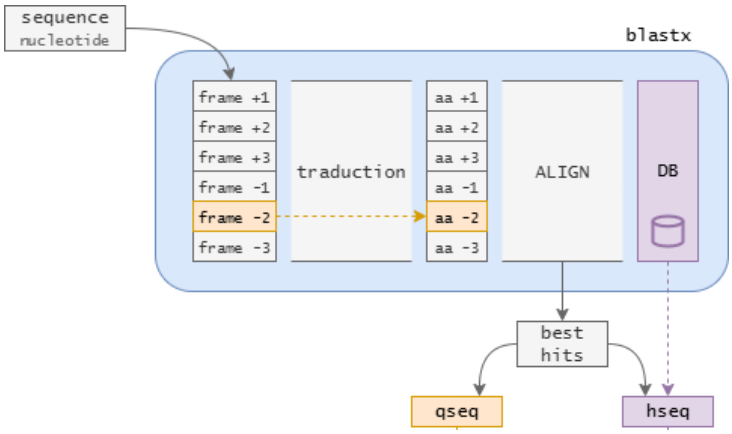

Figura 62. Diagrama blastx

Traducción

Este endpoint traduce la secuencia ingresada utilizando el marco de lectura del hit seleccionado.

Predicción de estructura con Neurosnap

Los endpoints siguientes inicializan la predicción, verifican el estado del proceso y obtienen la incertidumbre de los cinco modelos generados. Una vez completada la predicción, se puede descargar el PDB correspondiente a la predicción.

Descarga del modelo de referencia

Este endpoint utiliza una consulta adicional a UniProt para determinar si la estructura de referencia es experimental o predicha por AlphaFoldDB, arma la URL y luego descarga el PDB correspondiente.

Alineamiento

Finalmente, este último endpoint descarga tanto la predicción de Neurosnap como el modelo de referencia y los envía al servicio Python para realizar el alineamiento estructural. Devuelve la predicción alineada para que el frontend lo visualice.

En las Figuras 63 y 64 se puede visualizar el diagrama de procesos de la funcionalidad de Análisis estructural.

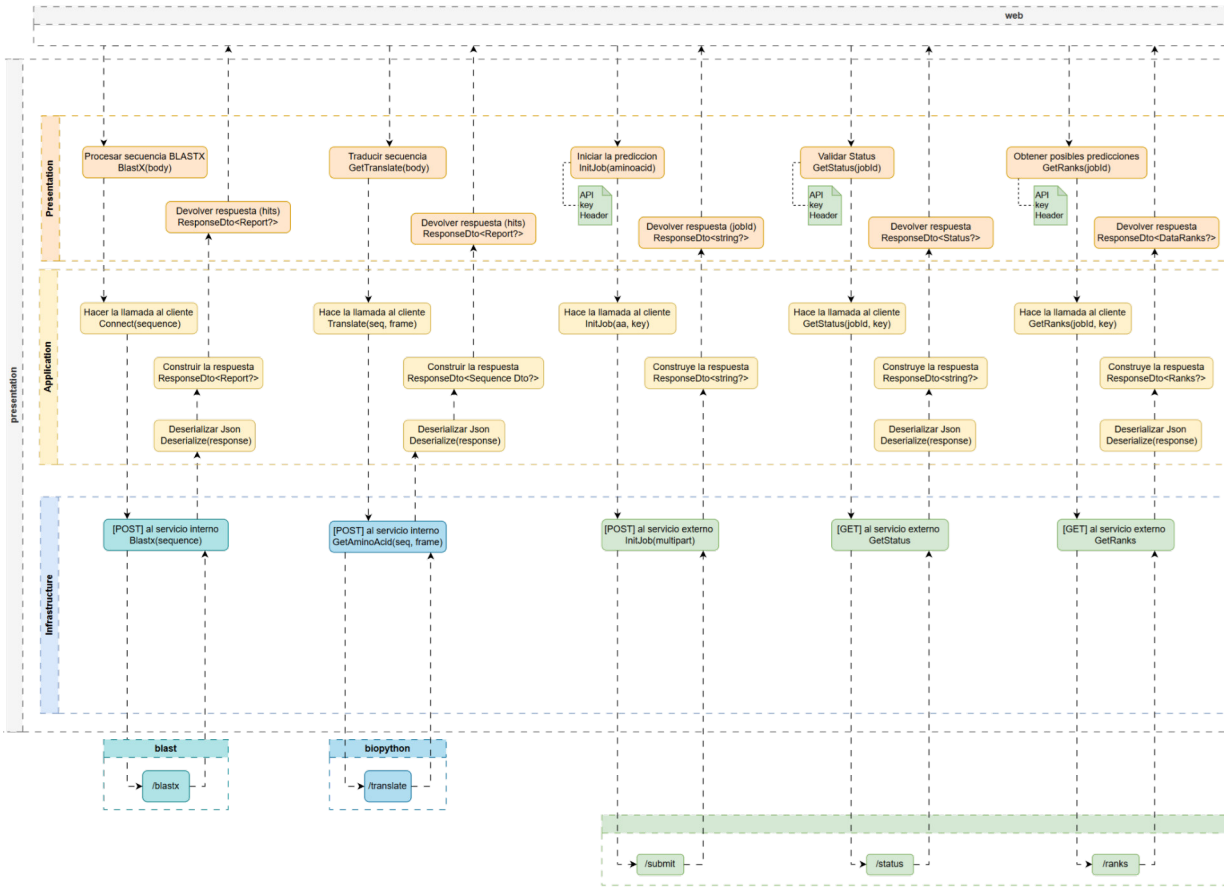

Figura 63. Diagrama de proceso de la funcionalidad de Análisis estructural (Parte 1).

Nota. Elaboración propia (link).

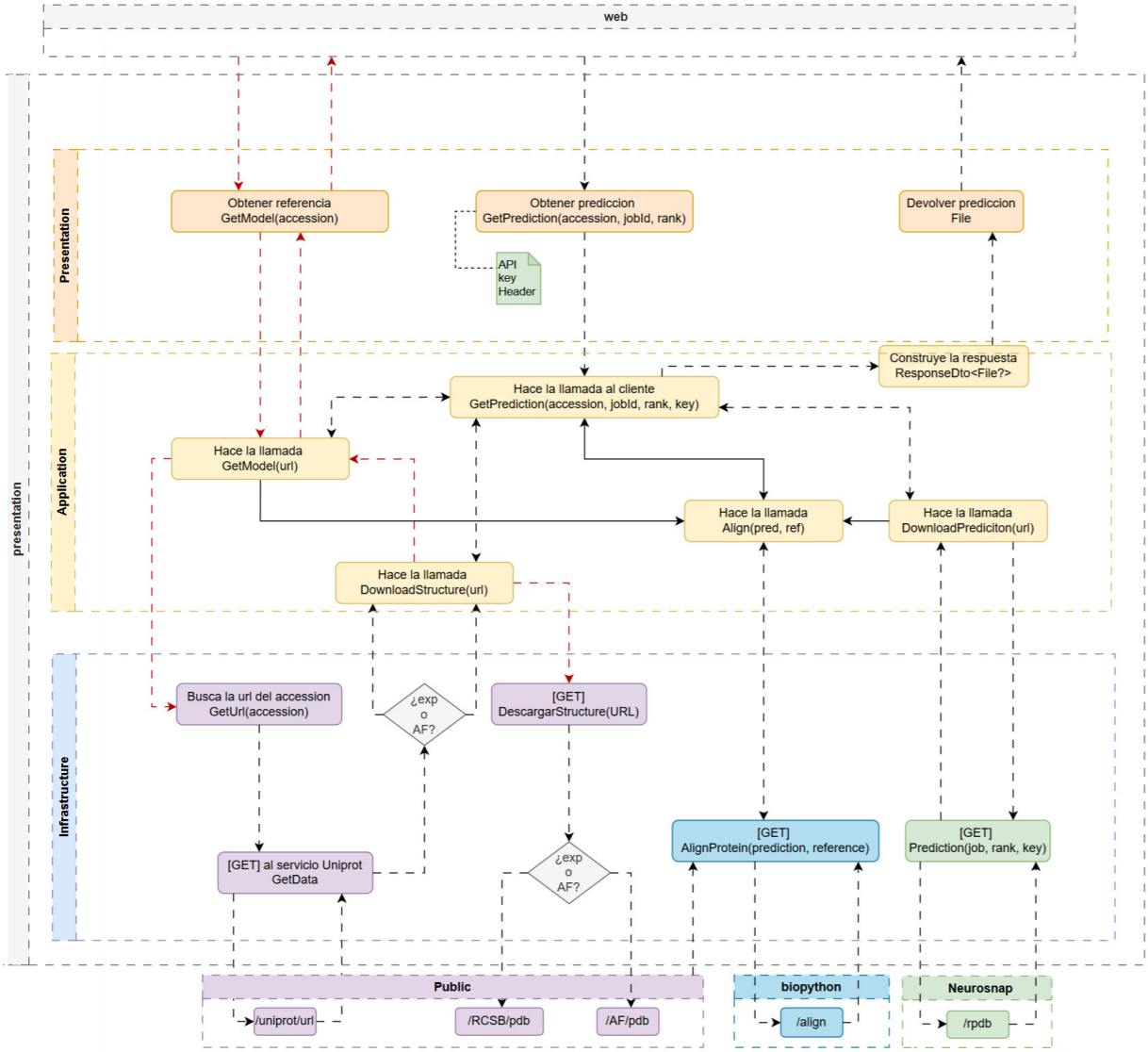

Figura 64. Diagrama de proceso de la funcionalidad de Análisis estructural (Parte 2).

Nota. Elaboración propia (link).

6.3.2 Detalles

Esta rama secundaria se centra en la obtención de información complementaria de genes y secuencias, incluyendo estadísticas, transformaciones básicas y visualización de datos. La funcionalidad permite explorar información genómica desde la vista de detalle y detalle full, acceder a secuencias, estadísticas y estructuras proteicas asociadas a cada UniProt ID.

La Figura 65 muestra el diagrama de flujo de esta funcionalidad:

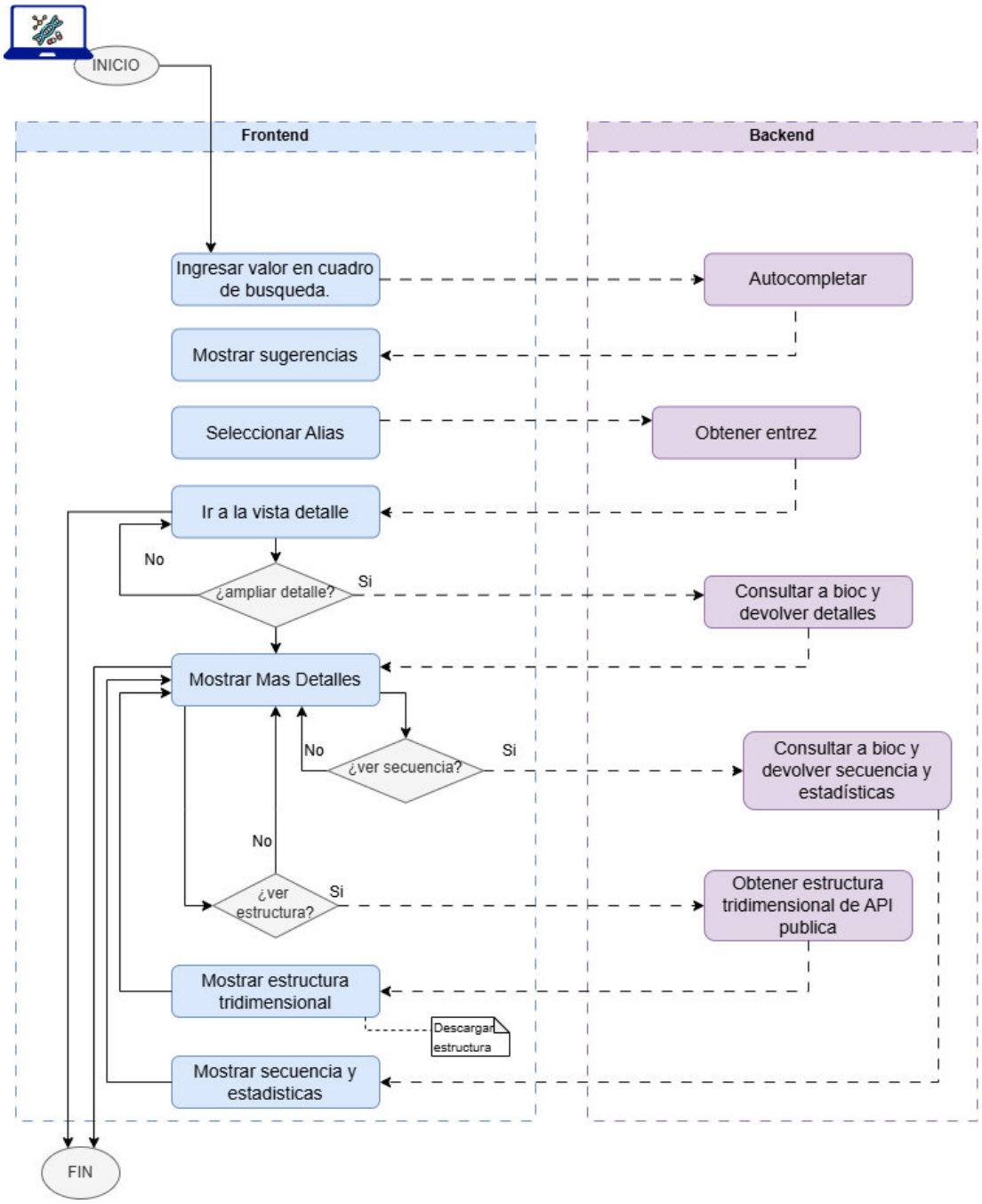

Figura 65. Diagrama de flujo de la funcionalidad de Detalle.

Nota. Elaboración propia (link).

6.4 Metodología de desarrollo

El desarrollo del proyecto siguió un enfoque incremental, por módulos y orientado a servicios. Esto permitió construir la solución paso a paso, integrando componentes independientes que posteriormente se conectaron en un flujo más amplio.

El proceso comenzó con la implementación de la API de Bioconductor, seguida de la creación de su respectivo Dockerfile para garantizar un entorno reproducible. A continuación, se desarrolló una pequeña API de prueba en .NET, con un endpoint básico de mensaje, junto con los primeros DTOs estandarizados necesarios para la comunicación eficiente entre servicios y el manejo consultas.

Luego se incorporó el frontend en React con TypeScript, configurado mediante Vite, y se trabajó en la integración de esos tres contenedores mediante docker-compose, lo cual permitió la comunicación desde el frontend hasta el servicio interno gestionado por .NET.

Durante esta etapa, se realizaron consultas a las APIs externas y se implementaron algunas utilidades simples.

Se agregó el contenedor BLAST para la búsqueda de secuencias similares a una dada y una vez resuelta la obtención del hit, Biopython para poder realizar alineamiento de estructuras. Así se completó la arquitectura de microservicios, cada módulo se desarrolló de manera independiente y luego se integró para conformar el flujo completo de análisis genómico y estructural.

6.5 Verificación

El diseño de la solución se verificó mediante pruebas integrales de los flujos funcionales implementados. Se realizaron pruebas manuales durante todo el desarrollo, primero a través de Swagger y luego evaluando la integración con el frontend, lo que permitió detectar puntos de mejora.

6.6 Dificultades

Durante el desarrollo se identificaron varias dificultades técnicas y de planificación:

6.5.1 Interoperabilidad

La integración de servicios desarrollados en R, Python y .NET presentó desafíos en la definición de los DTOs para la estandarización de la comunicación entre APIs. Inicialmente, los servicios *Bioc* y *Biopython* devolvían códigos de error que no permitían capturar excepciones en .NET. La solución fue modificar los servicios para que lanzaran excepciones propias y uniformes, permitiendo un manejo de errores centralizado en .NET.

6.5.2 Base de datos para BLAST

Inicialmente se utilizó PDB para las consultas de BLAST pero al ingresar una secuencia de cierta proteína que no poseía estructura experimental, el servicio devolvía hits que no tenían relación con la secuencia de entrada. Frente a esta dificultad se incorporó SwissProt como

base de datos más adecuada, evitando fallos al cargar proteínas sin estructura experimental asociada, se agregó una funcionalidad en .NET para consultar Uniprot y resolver la url que permitirá descargar la estructura de la proteína.

6.5.3 Predicción de estructuras de proteínas

Frente a la necesidad de una herramienta de predicción dentro de la aplicación, se evaluaron distintas alternativas. La implementación de *AlphaFold* de manera local (o en un cloud gratuito) resultaba inviable, dado que requiere alrededor de 3 TB de espacio en disco y GPU, además de recursos de cómputo significativos. Por otra parte, la ejecución en entornos como Colab, implicaba una pérdida de automatización del flujo de trabajo, ya que las predicciones debían descargarse manualmente y luego integrarse en la aplicación.

Finalmente, se optó por utilizar Neurosnap que, si bien ofrece créditos limitados mensuales para predicciones de proteínas, mantiene el flujo automatizado. Su utilización requería una *API Key* que, inicialmente, se almacenaba en el *backend*. Sin embargo, se implementó su gestión desde el frontend para que cada usuario controle su uso.

6.5.4 Despliegue en la nube

Se contempló inicialmente desplegar el proyecto en la nube, pero el consumo de recursos y la cantidad de contenedores no permiten esta opción con los créditos disponibles, así que se descartó y se optó por ejecutar los contenedores localmente.

6.5.5 Extensiones del navegador

Durante el desarrollo del frontend en React, la consola del navegador, se recomendó la instalación de la extensión. React DevTools para inspeccionar de manera más amigable el árbol de componentes. Sin embargo, se observó que al utilizar esta extensión en el navegador Brave, la aplicación se cerraba inesperadamente y se interrumpía la depuración del backend en .NET, por lo que se optó por visualizar la aplicación en Google Chrome.

Sección 7: Conclusiones

La experiencia permitió aplicar conceptos teóricos y prácticos adquiridos durante la carrera abarcando el ciclo de diseño, implementación y la entrega de un sistema funcional; y complementarlos con otros necesarios para el cumplimiento de los objetivos del proyecto.

Durante el desarrollo se incorporaron funcionalidades de autocompletado para sugerencias predictivas, la recuperación de detalles, la obtención de secuencias por cromosoma y rango con sus estadísticas básicas, el análisis BLASTX, la traducción y cálculo de reverso y complemento, y el alineamiento de secuencias y estructuras con su representación gráfica. En conjunto, estas capacidades cooperaron para formar las funcionalidades principales: el análisis estructural, la vista de detalle y otras utilidades que permitieron construir esta herramienta orientada al análisis bioinformático.

Como perspectiva a futuro, la aplicación ofrece líneas de mejora pendientes. En primer lugar, la incorporación de una base de datos simple para almacenar usuarios y contraseñas, para que cada usuario pueda manejar su cuenta desde una vista de *Usuarios* y vincularla con *Neurosnap,* y que cada cuenta permita asociar las consultas recientes a su perfil. También quedó pendiente la funcionalidad de descarga de informes o tablas en PDF, por ejemplo, el reporte del detalle del gen, la tabla de hits de BLASTX o el flujo de predicción. Otra mejora podría ser completar la vista de proteínas, donde el usuario pueda ingresar dos estructuras (por identificador o subiendo el archivo) para que el sistema las alinee y las represente gráficamente. En este sentido, en la visualización 3D se podrían sumar más elementos de interacción entre el usuario y la vista, como modificar parámetros visuales o colorear la predicción a medida que el mouse recorre secciones de la estructura primaria. Finalmente, se podría mejorar el autocompletado para incorporar búsquedas por nombre del gen.

El interactuar con personas provenientes de otras disciplinas representó para mí una oportunidad para ampliar mi perspectiva sobre el proyecto. Durante la cursada de Bioinformática surgió la oportunidad de empezar con un proyecto para la Práctica Profesional Supervisada. Ramiro se puso en contacto con Eduardo para que nos sugiriera posibles líneas de trabajo. Ahí surgió la idea de analizar las secuencias de los originarios para estudiar el problema del raquitismo, y luego pensamos qué construir, en el marco del proyecto, para contribuir a esta problemática. En esta etapa también se incorporaron Ariel, el nuevo docente de Bioinformática, que fue una guía constante, y Lucas, cuya orientación durante la materia Proyecto de Software fue clave para seleccionar un proyecto de desarrollo con este enfoque. Con estas incorporaciones terminamos de conformar el equipo de trabajo.

Cuando empezamos no tenía claro lo que había que hacer así que comencé probando herramientas de bioconductor e integrándolas con .NET y el *frontend.* Cuando nos reunimos a ver los avances, Eduardo mencionó que *“había que alinear con blast y comparar”.* En ese momento no entendí cómo íbamos a hacerlo pero esa frase se convirtió en el eje central de la aplicación. A partir de ahí la aplicación fue creciendo por etapas: incorporamos *blast* para analizar los hits y el marco de lectura, luego fue necesario sumar una herramienta de predicción y, por último, surgió la necesidad de alinear con *biopython* para mostrar los resultados. Este proceso permitió entender el alcance real del proyecto y transformó mi visión inicial de lo que sería una aplicación web en bioinformática.

En conjunto, el trabajo resultó una experiencia formativa y la interacción permitió comprender como una solución informática puede contribuir a un problema científico real. Además, me aportó una visión más amplia sobre el rol de la ingeniería en informática en entornos interdisciplinarios y dejó una base abierta para futuras mejoras.

Bibliografía

3Dmol.js. (s. f.). *3Dmol.js*. https://www.npmjs.com/package/3dmol

Academia Lab. (2025). BLAST (biotecnología). Enciclopedia. Revisado el 9 de diciembre del 2025. https://academia-lab.com/enciclopedia/blast-biotecnologia/

ADN-Institut. (s. f.). *Qué son los genes, cómo se expresan*. https://www.adninstitut.com/que-son-los-genes-como-se-expresan-n-66-es

Ahern, E., Rajagopal, K., & Tan, R. (s. f.). *Estructura y función – Aminoácidos*. LibreTexts. http://biochem.science.oregonstate.edu/content/biochemistry-free-and-easy

Ahern, E., Rajagopal, K., & Tan, R. (s. f.). *Estructura y función – Proteínas I*. LibreTexts. http://biochem.science.oregonstate.edu/content/biochemistry-free-and-easy

Alphafold. (s. f.). *Frequently asked questions*. https://alphafold.ebi.ac.uk/faq

AlphaFold. (s. f.). *Home*. https://alphafold.ebi.ac.uk/ ebi.ac.uk

Bioconductor. (s. f.). *AnnotationDbi: Annotation Database Interface*. https://bioconductor.org/packages/release/bioc/html/AnnotationDbi.html

Bioconductor. (s. f.). *Bioconductor*. https://www.bioconductor.org/

Bioconductor. (s. f.). *Biostrings: Efficient manipulation of biological strings*. https://bioconductor.org/packages/release/bioc/html/Biostrings.html

Bioconductor. (s. f.). *BSgenome: Full genome sequences for R*. https://bioconductor.org/packages/release/bioc/html/BSgenome.html

Bioconductor. (s. f.). *org.Hs.eg.db: Genome wide annotation for Human*. https://bioconductor.org/packages/release/data/annotation/html/org.Hs.eg.db.html

Bioconductor. (s. f.). *TxDb.Hsapiens.UCSC.hg38.knownGene: Transcript database*. https://www.bioconductor.org/packages/release/data/annotation/html/TxDb.Hsapiens.UCSC.hg38.knownGene.html

BioPython. (s. f.). *Biopython*. https://biopython.org/

DeepMind. (2024). *AlphaFold* [Repositorio GitHub]. GitHub. https://github.com/google-deepmind/alphafold

Docker, Inc. (s. f.). *Docker*. https://www.docker.com/

Docker Hub. (s. f.). *Docker Hub*. https://hub.docker.com/

EBI Training. (s. f.). *pLDDT – Understanding local confidence in AlphaFold predicted structures*. https://www.ebi.ac.uk/training/online/courses/alphafold/inputs-and-outputs/evaluating-alphafolds-predicted-structures-using-confidence-scores/plddt-understanding-local-confidence/

Guyton, A. C., & Hall, J. E. (2021). *Tratado de fisiología médica* (14.ª ed.). Elsevier.

Labster. (s. f.). *Aminoácidos*. https://theory.labster.com/es/amino-acids/

Labster. (s. f.). *Estructura proteica*. https://theory.labster.com/es/protein-structure/

MedlinePlus. (s. f.). *Ubicación de los genes*. https://medlineplus.gov/spanish/genetica/entender/comofuncionangenes/genubicacion/

Nature. (2021). *Highly accurate protein structure prediction with AlphaFold*. https://www.nature.com/articles/s41586-021-03819-2

NCBI. (s. f.). *National Center for Biotechnology Information*. https://www.ncbi.nlm.nih.gov/

Neurosnap. (s. f.). *AlphaFold2 service*. https://neurosnap.ai/service/AlphaFold2

Neurosnap. (s. f.). *Overview*. https://neurosnap.ai/overview

NIH. (s. f.). *National Institutes of Health*. https://www.nih.gov/

Plotly Technologies Inc. (s. f.). React. Plotly. https://plotly.com/javascript/react/

RCSB Protein Data Bank. (s. f.). *RCSB PDB*. https://www.rcsb.org/

F. D. Saraví, J. H. Wilches Visbal. (2022). La accidentada historia del descubrimiento de la vitamina D, en su primer centenario. *Osteología, 18(3): 157-168.* https://ojs.osteologia.org.ar/ojs33010/index.php/osteologia/article/view/67/49

UniProt Consortium. (s. f.). *UniProt*. https://www.uniprot.org/ UniProt

Visual Studio Code. (s. f.). *Visual Studio Code*. https://code.visualstudio.com/

Visual Studio. (s. f.). *Visual Studio*. https://visualstudio.microsoft.com/es/