2025

Desarrollo de Software - Mutantes

ALUMNA: BRENDA ALCOBA

COMISIÓN: 3K9
LEGAJO: 50756

INTRODUCCIÓN

El presente trabajo consiste en el desarrollo de un sistema capaz de identificar si una secuencia de ADN pertenece a un humano o a un mutante, utilizando técnicas de análisis basadas en patrones repetitivos. El proyecto fue realizado como parte de la evaluación de la materia, integrando conceptos fundamentales de ingeniería de software, arquitectura de aplicaciones, pruebas automatizadas y buenas prácticas de desarrollo.

Para resolver el problema propuesto se implementó una **API REST desarrollada con Spring Boot**, que recibe matrices de ADN en formato JSON y determina su clasificación según reglas específicas. A lo largo del proyecto se aplicó una división clara por capas (controlador, servicio, lógica de detección y persistencia), manteniendo una arquitectura limpia y fácil de extender.

Además del análisis del ADN, el sistema incorpora mecanismos de validación, manejo formal de errores, registro de resultados en base de datos y una estrategia de optimización mediante almacenamiento de hashes para evitar procesamiento duplicado. También se desarrollaron **tests unitarios e integrales**, con el fin de asegurar la confiabilidad de cada componente y cumplir con los criterios de evaluación.

Arquitectura del Proyecto

El proyecto se organizó siguiendo una arquitectura por capas para mantener el código ordenado y fácil de entender. Cada parte del sistema tiene una función clara y trabaja en conjunto para resolver si un ADN es mutante o no.

1. Controller (Entrada de la API)

Aquí se reciben las solicitudes del cliente, como el POST **/mutant** y el GET **/stats**.
Esta capa valida que el ADN venga en el formato correcto y luego delega todo el trabajo al servicio.
Su función principal es coordinar el flujo y devolver la respuesta adecuada (200, 403 o 400).

2. Service (Lógica de negocio)

En esta parte se toma la decisión importante:

- si el ADN ya existe, se usa el resultado guardado,
- si no, se analiza con el detector.

El servicio también guarda la información en la base y calcula las estadísticas del sistema mediante `StatsService`.

3. Detector (Algoritmo)

Es la parte del proyecto que realmente revisa si el ADN tiene secuencias mutantes.
El algoritmo busca patrones en distintas direcciones (horizontal, vertical y diagonales) y devuelve si es mutante o no.
Está separado para que sea más fácil de testear y mantener.

4. Repository (Persistencia)

Esta capa maneja la base de datos usando Spring Data JPA.
Guarda cada ADN con su hash, la fecha y si era mutante o humano.
También permite consultar estadísticas sin escribir SQL manual.

5. DTOs y Validaciones

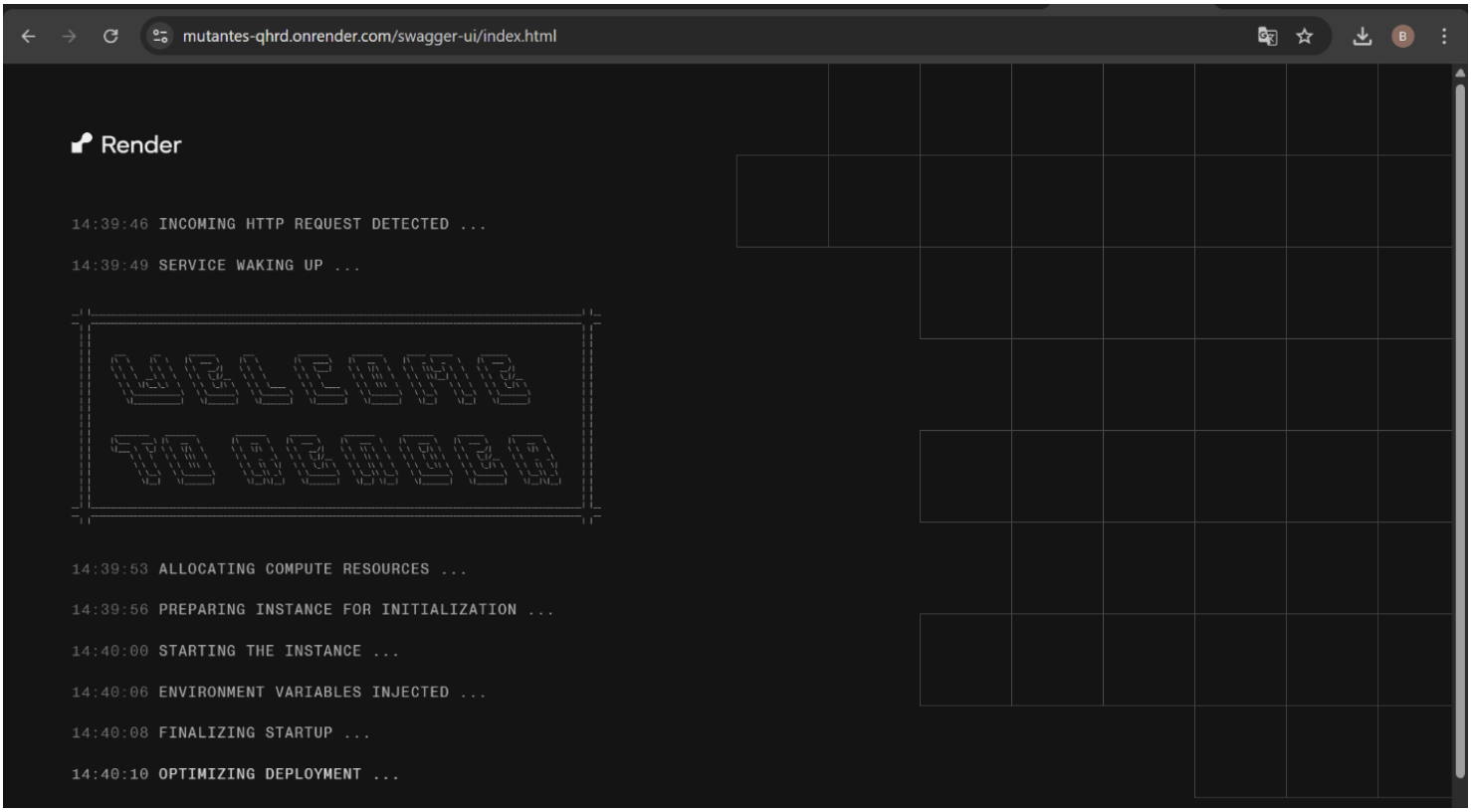
Se usan objetos simples (DTOs) para recibir y enviar datos de forma limpia.
Además, hay validaciones que aseguran que el ADN sea una matriz NxN y que sólo contenga letras válidas.

Despliegue de la Aplicación

Luego de haber construido la estructura del proyecto y desarrollado todas las capas necesarias (controlador, servicios, repositorio y algoritmo de detección), realicé diferentes pruebas locales para verificar que la API funcionara correctamente.
Una vez validado el funcionamiento mediante los tests unitarios, de integración y las pruebas manuales, procedí a publicar la aplicación en la nube utilizando **Render**, una plataforma que permite desplegar proyectos de Spring Boot de manera sencilla.

URL DEL DEPLOY: https://mutantes-qhrd.onrender.com/swagger-ui/index.html

Al ingresar, Render genera automáticamente la documentación interactiva de Swagger, desde donde es posible probar los endpoints sin necesidad de ejecutar el proyecto localmente. A continuación se muestran algunas capturas que ilustran el funcionamiento de la API desplegada y las pruebas realizadas directamente desde el entorno online.



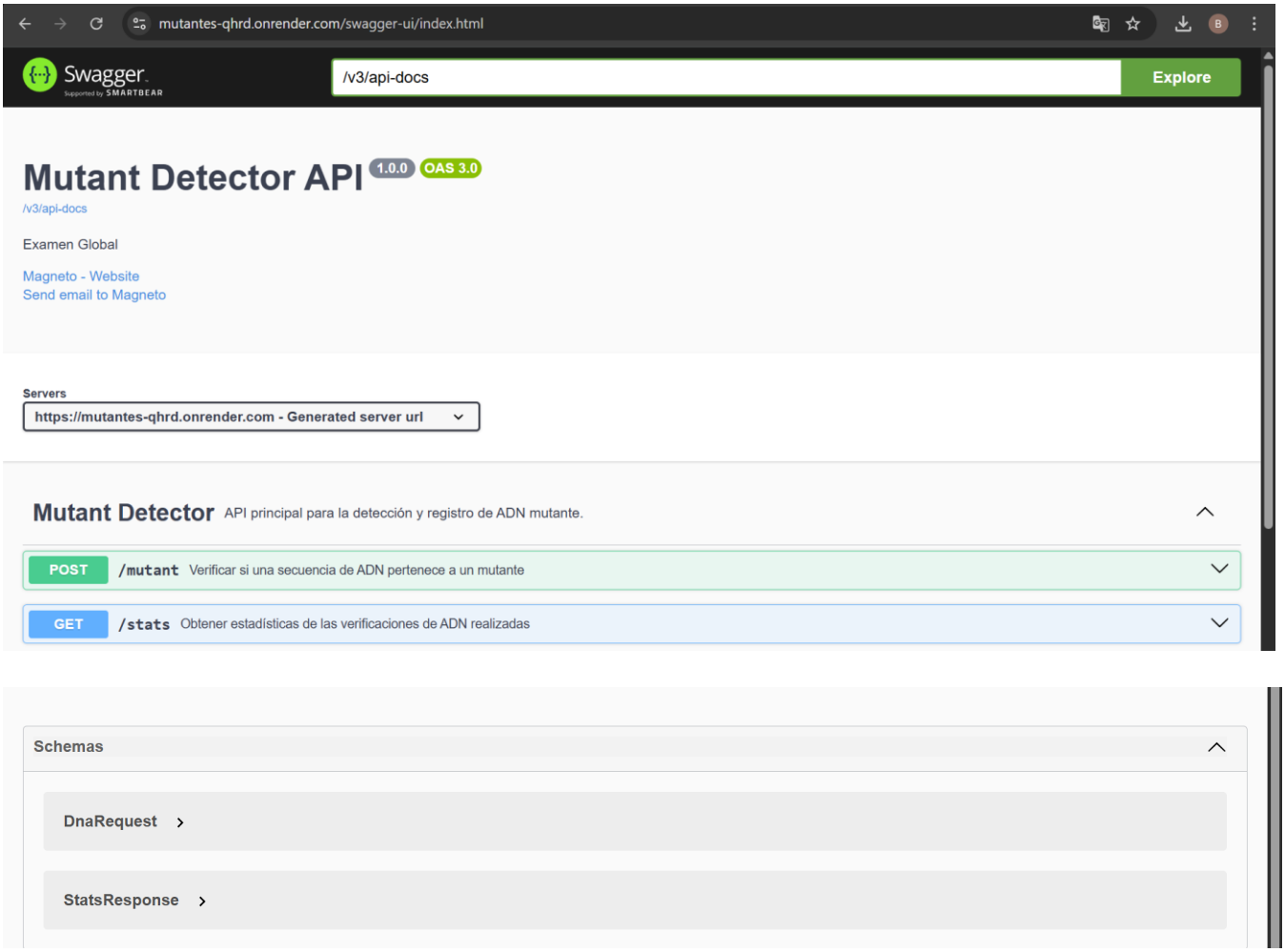


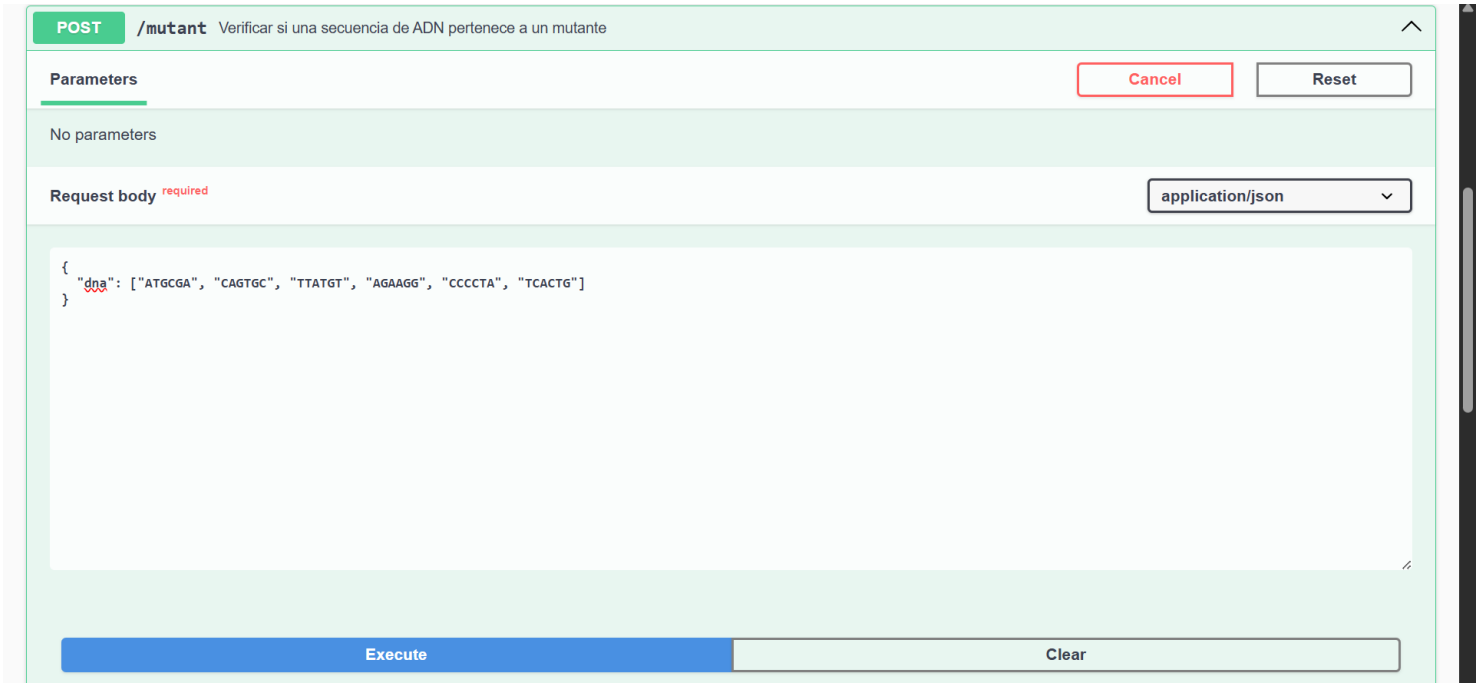
Figura 1. *Interfaz inicial de Swagger generada automáticamente en el entorno de producción desplegado en Render.*

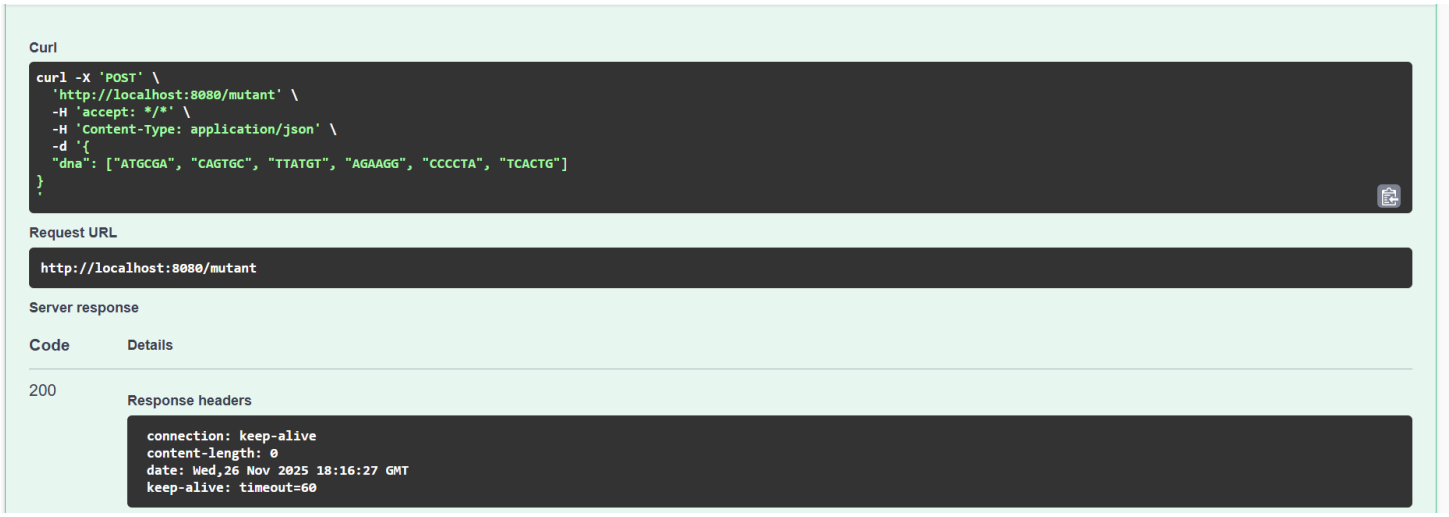
Endpoint POST /mutant

Code	Description
200	OK. El ADN es mutante (contiene más de una secuencia de 4 bases iguales).
400	BAD REQUEST. La secuencia de ADN es inválida (no es NxN o contiene caracteres no permitidos).
403	FORBIDDEN. El ADN no es mutante (contiene 0 o 1 secuencia).

Caso 1: MUTANTE

```
{
  "dna": ["ATGCGA", "CAGTGC", "TTATGT", "AGAAGG", "CCCCTA", "TCACTG"]
}
```



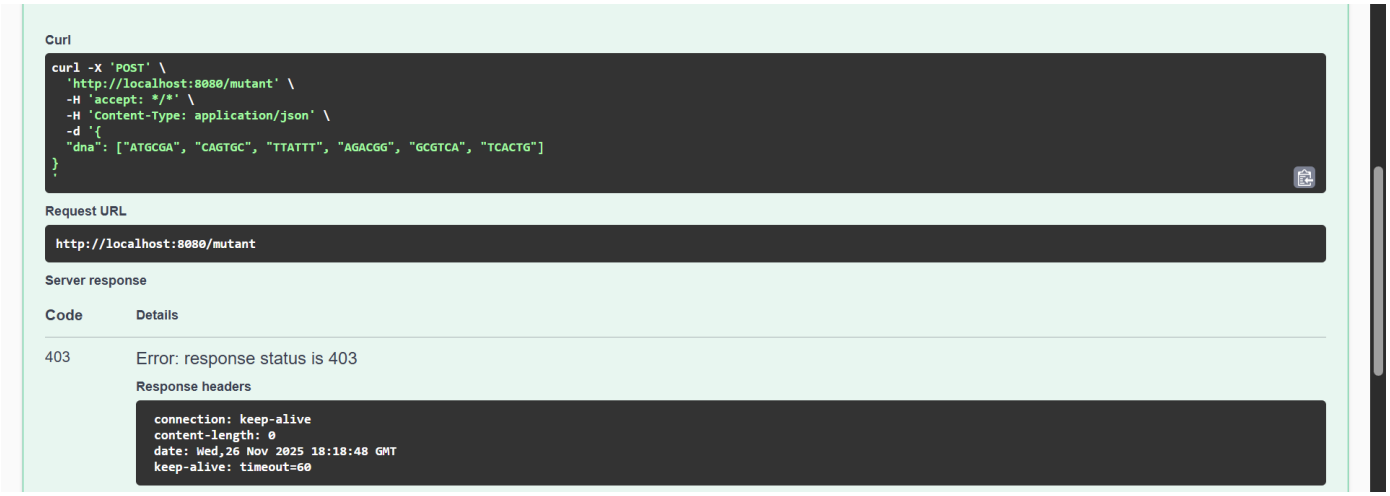


Caso 2: HUMANO

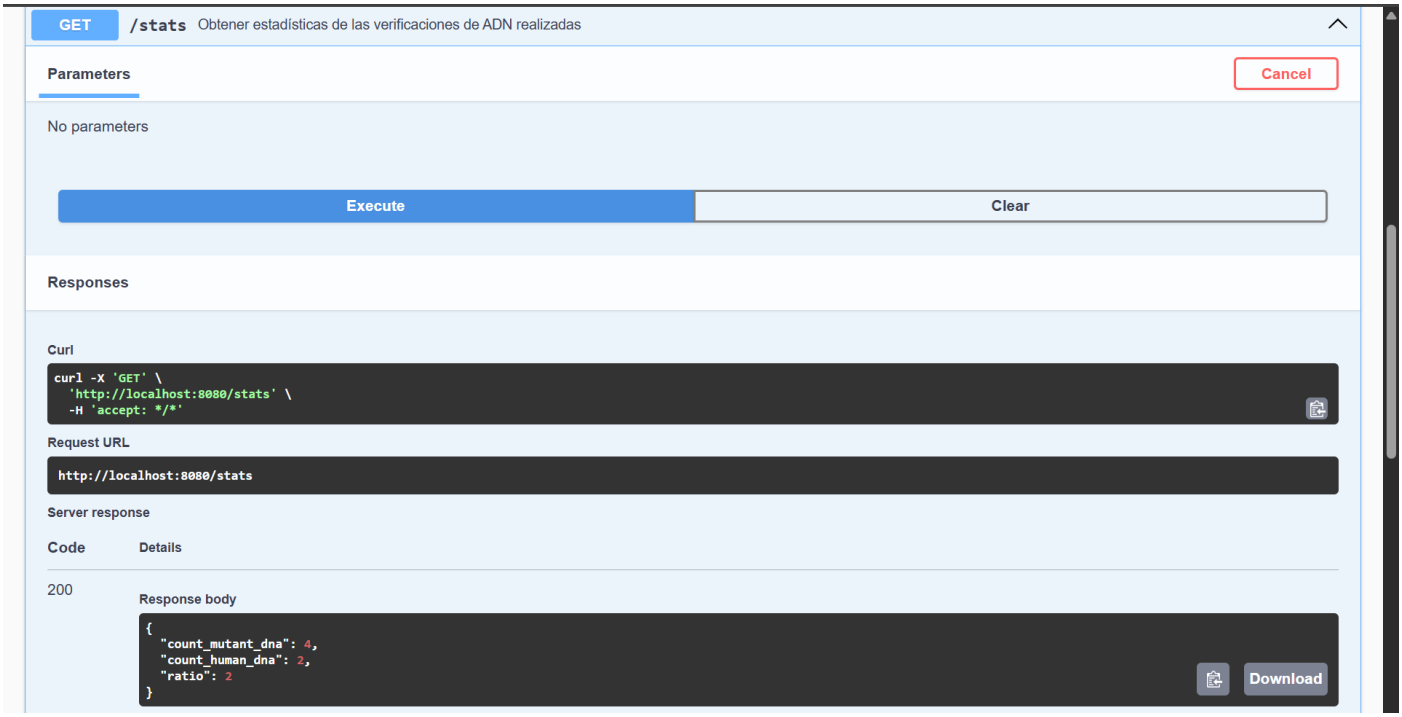
{

"dna": ["ATGCGA", "CAGTGC", "TTATTT", "AGACGG", "GCGTCA", "TCACTG"]

}



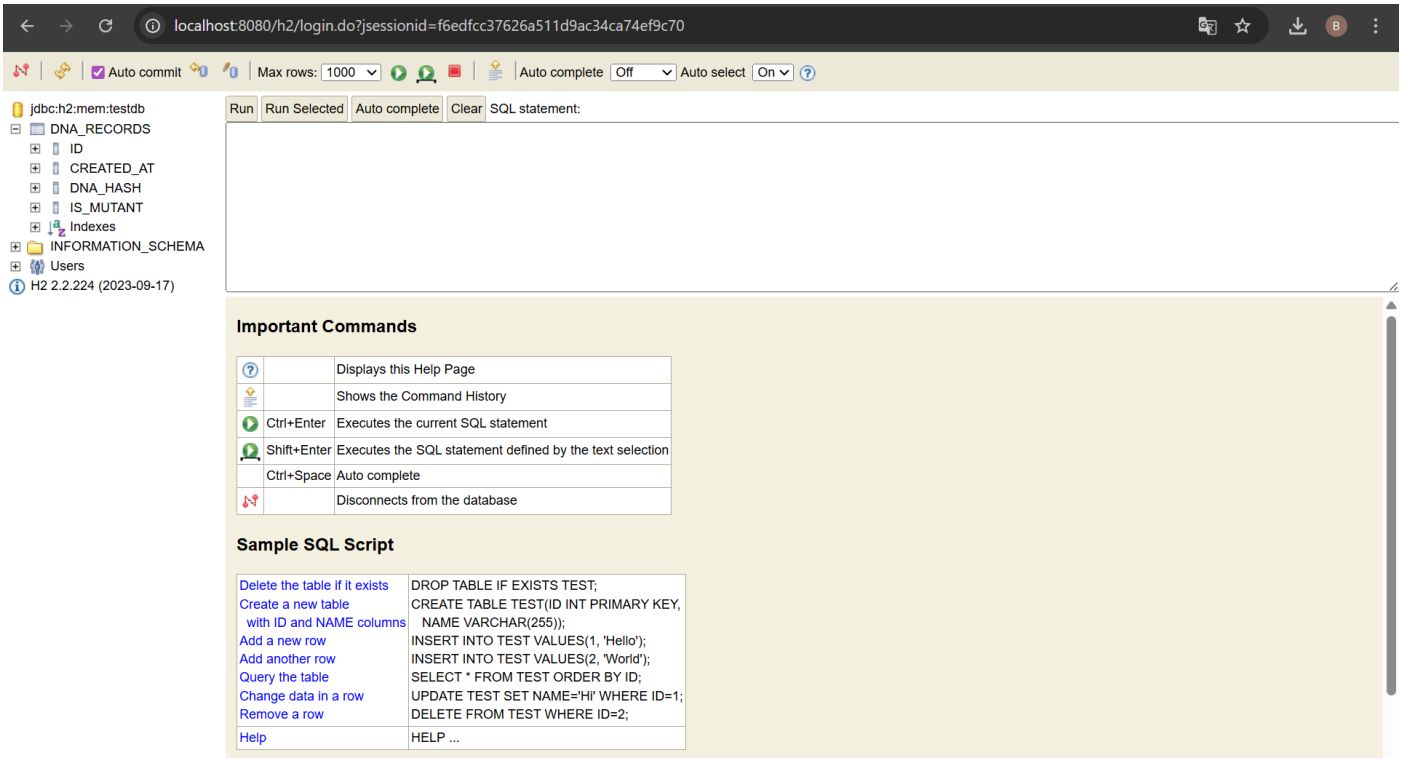
GET /stats



En esta figura se observa el funcionamiento del endpoint /stats, el cual permite consultar las estadísticas generadas a partir de todas las verificaciones de ADN realizadas previamente. La API devuelve un JSON con el total de ADN mutante detectado, el total de ADN humano y el ratio entre ambos valores. Esta consulta permite validar que la aplicación almacena los resultados y los expone correctamente a través del servicio REST.

H2

Además de los endpoints expuestos por la API, la aplicación utiliza una base de datos en memoria H2 para almacenar los ADN analizados. Durante el desarrollo verifiqué la correcta persistencia de los datos ingresando a la consola H2, lo que permitió validar la estructura de la tabla y los registros generados durante las pruebas.



Conclusión

Este proyecto me permitió integrar varios conceptos vistos durante la cursada, especialmente arquitectura por capas, validaciones, pruebas y despliegue en la nube.
También pude entender mejor cómo se organiza una API real y cómo interactúan sus componentes internos.

Aunque fue un desafío, resultó una experiencia valiosa porque me acercó más al desarrollo backend y a herramientas prácticas como Spring Boot, JPA, Swagger y Render. Considero que fue una actividad útil, que me permitió mejorar mi lógica, mi manera de estructurar código y mi seguridad en el desarrollo de servicios web.