GLOBAL DESARROLLO DE SOFTWARE

MUTANT DETECTOR

Alumno: **Facundo Nicolas Carrillo Fugazzotto**

Materia: Desarrollo de Software

Fecha: Noviembre 2025

1. Enlaces del Proyecto

A continuación se detallan los enlaces al repositorio de código fuente y a la API desplegada en la nube para su corrección.

- **Repositorio GitHub (Código Fuente):** https://github.com/FacundoCarrillo/mutantDetector.git
- **URL de la API (Despliegue en Render):** https://mutantdetector-svn7.onrender.com

2. Instrucciones Rápidas de Prueba

La API se encuentra activa y escuchando peticiones. Se recomienda utilizar Postman

- **Detectar Mutante (POST):**

Endpoint: `https://mutantdetector-svn7.onrender.com/mutant/`

Body (JSON):
JSON

```
{

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

}
```

  - Respuesta esperada: `200 OK`

SI NO ES MUTANTE

  - Respuesta esperada: `403 forbidden`

- **Ver Estadísticas (GET):**

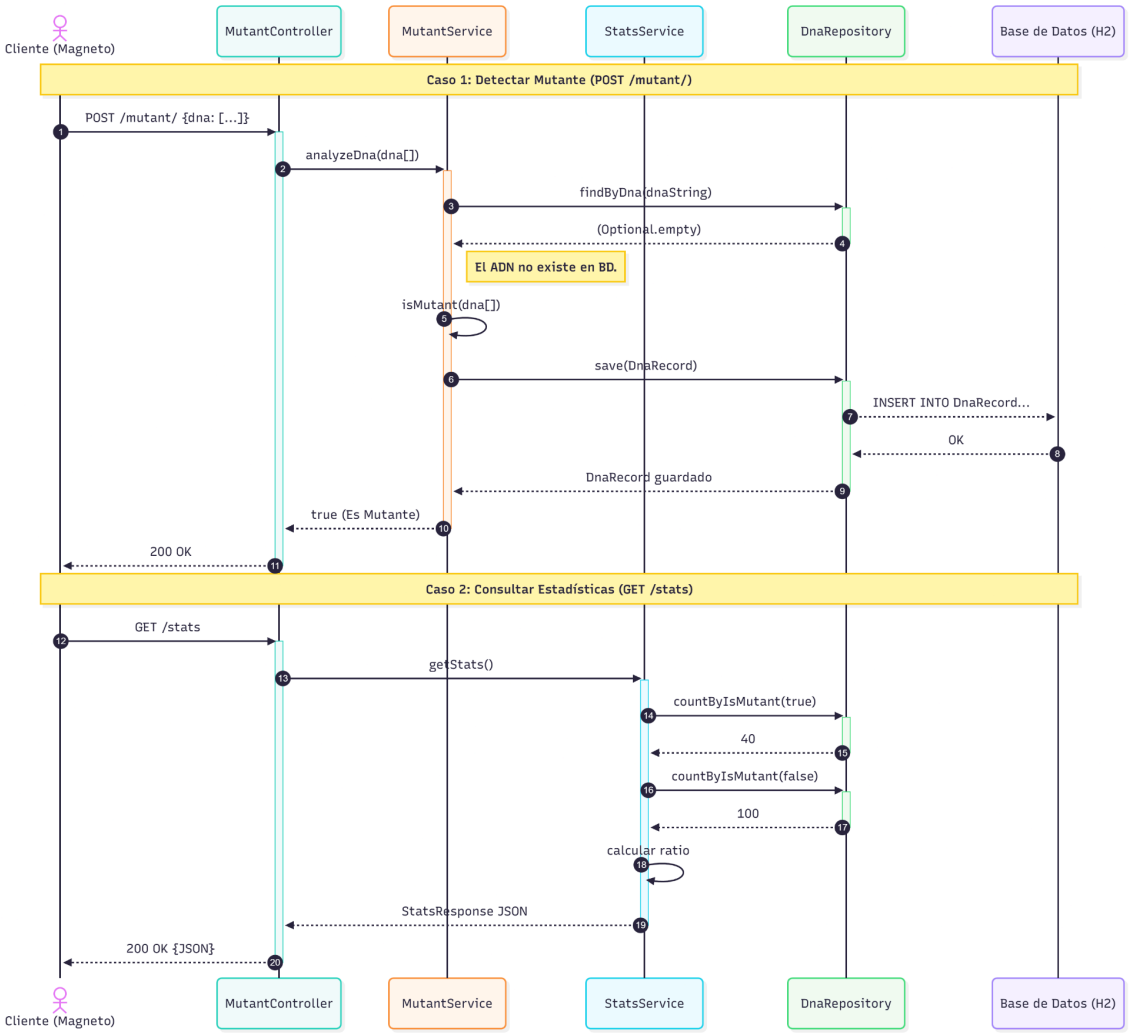
Endpoint: `https://mutantdetector-svn7.onrender.com/stats`

  - Respuesta esperada: `200 OK`


```
{
    "count_mutant_dna": 3,
    "count_human_dna": 2,
    "ratio": 1.5
}
```

3. Diagrama de Secuencia

El siguiente diagrama ilustra el flujo de la información desde la petición del cliente (Magneto) hasta la persistencia en la base de datos H2 y la respuesta de la API.



Nota: Se implementó una arquitectura en capas (Controller, Service, Repository) para asegurar la escalabilidad y mantenibilidad del código.

4. Reporte de Cobertura de Tests (Code Coverage)

Se han desarrollado tests unitarios y de integración utilizando **JUnit 5**, **Mockito** y **Spring Boot Test**. Se ha cumplido con el requisito de superar el 80% de cobertura de código.

- **Cobertura Global:** 80.5%

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %
all classes	100% (7/7)	76% (19/25)	75,9% (44/58)	80,5% (62/77)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
com.mutantes.mutantdetector	100% (1/1)	50% (1/2)		50% (1/2)
com.mutantes.mutantdetector.controller	100% (1/1)	100% (3/3)	100% (2/2)	100% (8/8)
com.mutantes.mutantdetector.dto	100% (2/2)	77,8% (7/9)		77,8% (7/9)
com.mutantes.mutantdetector.model	100% (1/1)	60% (3/5)		60% (3/5)
com.mutantes.mutantdetector.service	100% (2/2)	83,3% (5/6)	75% (42/56)	81,1% (43/53)

generated on 2025-11-26 00:26

5. Notas de Implementación y Desafíos

Durante el desarrollo del proyecto se tomaron decisiones técnicas específicas para cumplir con los requerimientos de eficiencia y robustez:

1. **Configuración de Entorno (Java 17 & Lombok):** Se realizó una configuración manual en el `pom.xml` (maven-compiler-plugin) para asegurar la compatibilidad correcta entre Java 17 y la librería Lombok (versión 1.18.30), garantizando que el procesado de anotaciones funcione correctamente durante la compilación.
2. **Eficiencia del Algoritmo (Big O):** El algoritmo de detección de mutantes implementa una estrategia de "Early Return" (retorno temprano). El sistema deja de recorrer la matriz inmediatamente después de encontrar la segunda secuencia de ADN mutante, optimizando el tiempo de respuesta y el uso de recursos, especialmente bajo alta concurrencia.
3. **Persistencia (Base de Datos):** Se utilizó una base de datos en memoria H2 con JPA. Se configuró la entidad `DnaRecord` con un índice único en la secuencia de ADN para evitar duplicados y consultas innecesarias.
4. **Despliegue con Docker:** Para asegurar que la aplicación corra en cualquier entorno, se creó un `Dockerfile` multietapa (Multi-stage build) que compila el proyecto con Maven y luego genera una imagen ligera con Eclipse Temurin JRE para su ejecución en producción (Render).