

## Informe del Proyecto – Detectando Mutantes

**Alumno:** Juan Gallardo

**Carrera:** Ingeniería en Sistemas – UTN FRM

**Tecnologías:** Java 17, Spring Boot 3.5, H2, JPA, SpringDoc, JUnit 5, Mockito, Jacoco

**Fecha:** 2025

---

### 1. Introducción

El proyecto implementa una API REST capaz de analizar secuencias de ADN y determinar si corresponden a un **mutante**.

Una persona es considerada mutante si su ADN contiene **al menos dos secuencias de cuatro letras iguales consecutivas** en alguna de las siguientes direcciones:

- Horizontal
- Vertical
- Diagonal
- Diagonal inversa

Solo se aceptan cadenas formadas por los caracteres: **A, T, C, G**.

Además, la API expone estadísticas globales y evita procesar ADN duplicado utilizando **hash SHA-256** para identificar secuencias previamente analizadas.

---

### 2. Arquitectura General

La aplicación sigue una arquitectura de tres capas:

#### ✓ Controlador (MutantController)

Gestiona los endpoints /mutant y /stats.

#### ✓ Servicio (MutantService, StatsService)

Implementa la lógica de validación, detección, hashing y persistencia.

#### ✓ Repositorio (DnaRecordRepository)

Acceso a la base de datos H2.

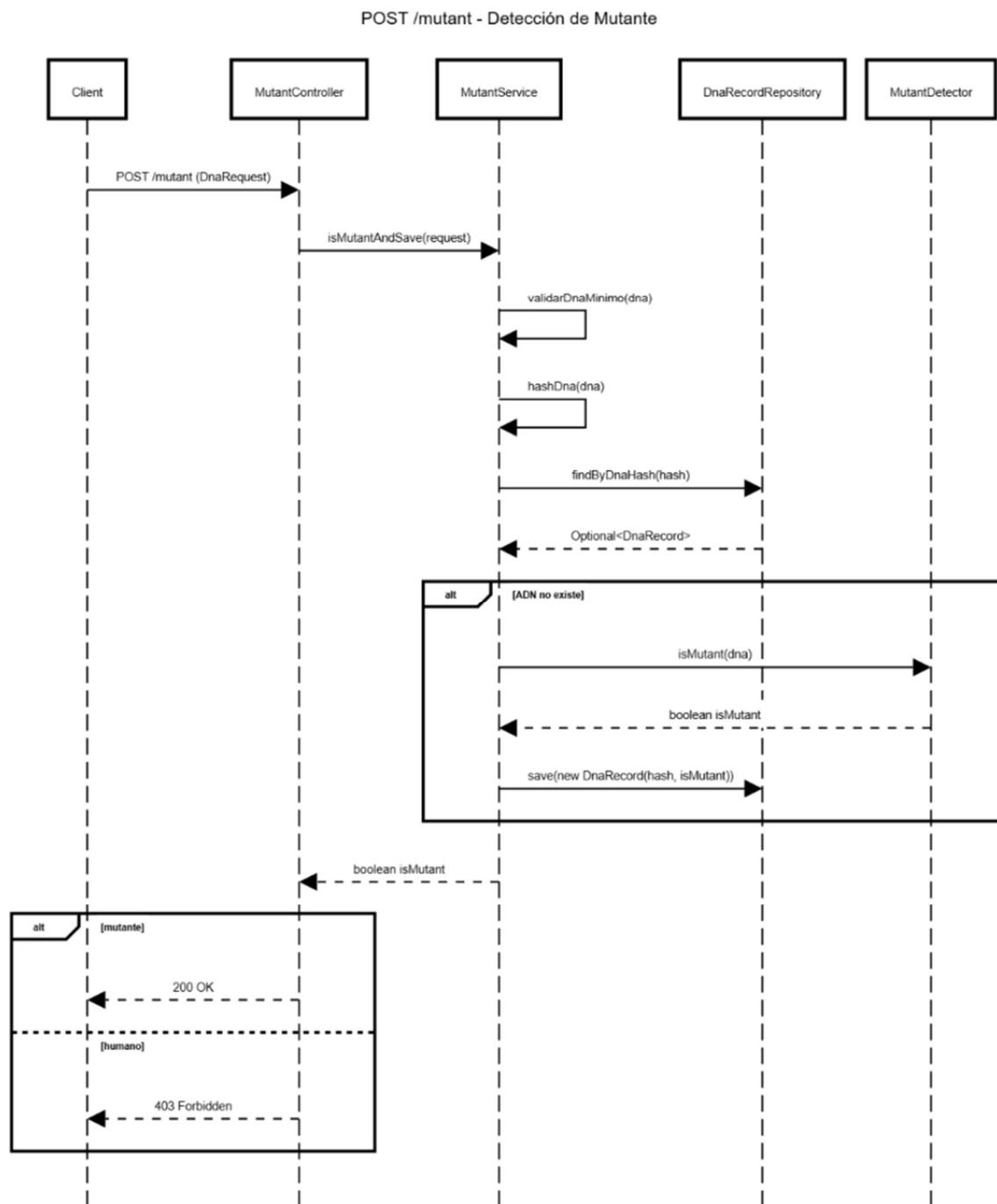
#### ✓ Detector (MutantDetector)

Algoritmo  $O(N^2)$  de detección de secuencias.

### ✓ Manejador Global de Errores (GlobalExceptionHandler)

Unifica respuestas de error con formato consistente.

### 3. Endpoint POST /mutant - Diagrama de Secuencia

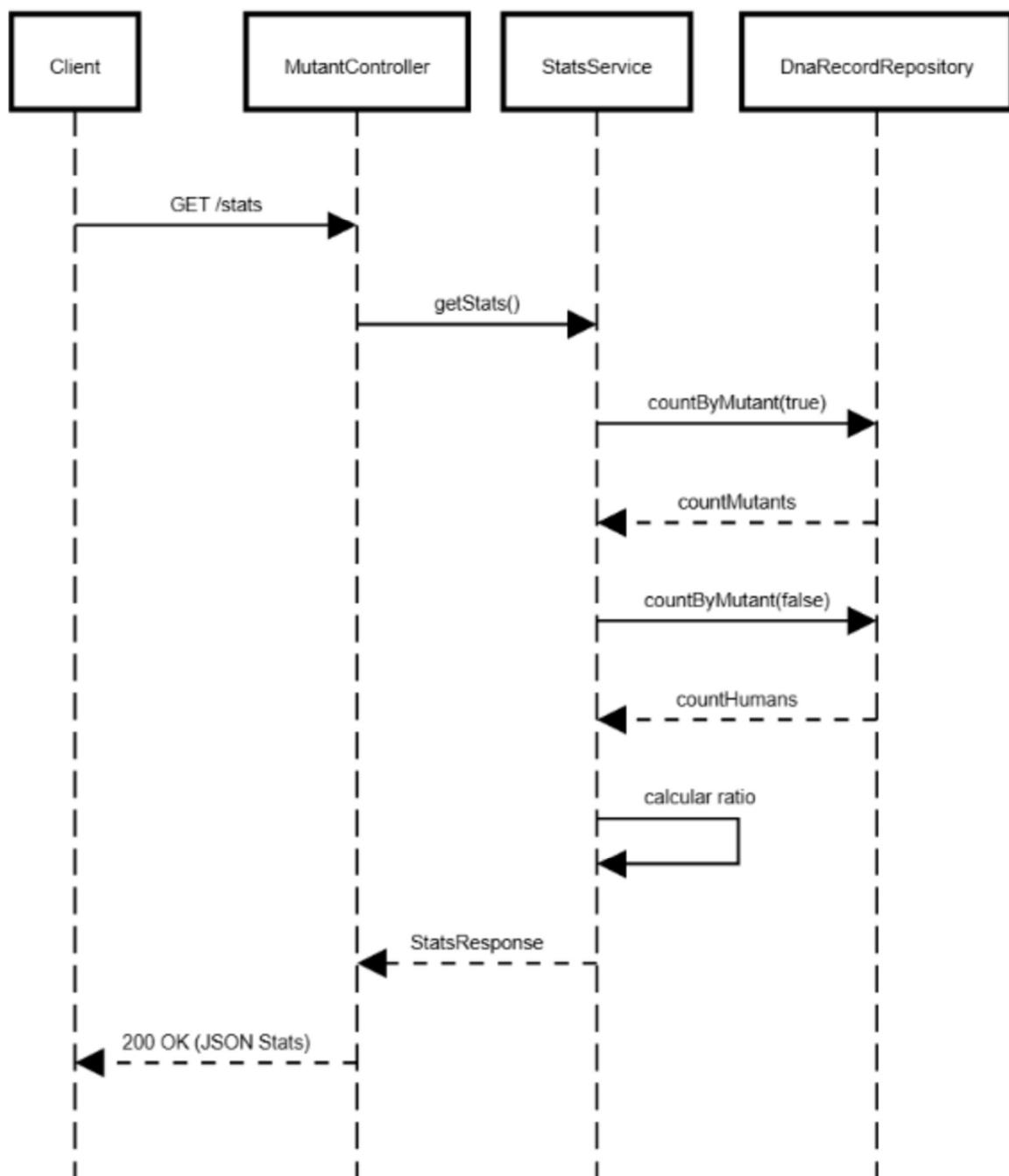


Descripción del flujo:

1. El cliente envía un DnaRequest.
  2. El controlador invoca al servicio.
  3. El servicio valida el ADN y genera el hash.
  4. Si el hash existe, retorna el resultado previo.
  5. Si no existe, llama al MutantDetector.
  6. Guarda el resultado en la base.
  7. Devuelve:
    - o **200 OK** si es mutante
    - o **403 Forbidden** si es humano
- 

#### 4. Endpoint GET /stats - Diagrama de Secuencia

### GET /stats - Estadísticas



Descripción del flujo:

1. El cliente consulta /stats.
2. El controlador invoca StatsService.
3. Se cuentan mutantes y humanos desde la base.

4. Se calcula el ratio.

5. Retorna un JSON:

```
{  
  "count_mutant_dna": 40,  
  "count_human_dna": 100,  
  "ratio": 0.4  
}
```

---

## 5. Documentación Swagger – SpringDoc OpenAPI

The screenshot shows a web browser displaying the Swagger UI at [localhost:8080/swagger-ui/index.html#/](http://localhost:8080/swagger-ui/index.html#/). The title bar indicates the URL and the page title is "v3/api-docs". The main content area is titled "OpenAPI definition v6 OAS 3.0" and shows the "Mutantes" section. Under "Servers", it lists "http://localhost:8080 - Generated server url". The "Mutantes" section contains two operations: "POST /mutant" (green button) which "Verifica si un ADN pertenece a un mutante", and "GET /stats" (blue button) which "Obtene estadisticas de mutantes". Below these, under "Schemas", there are definitions for "DnaRequest" and "StatsResponse".

---

## 6. Test Unitarios

Se desarrollaron pruebas unitarias y de integración con:

- **JUnit 5**
- **Mockito**
- **Spring Boot Test**
- **Jacoco** para cobertura

## ✓ MutantDetectorTest

Prueba detección horizontal, vertical, diagonal, inversa, casos borde y matrices grandes.

## ✓ MutantServiceTest

- Guarda mutantes
- Guarda humanos
- Evita duplicados
- Verifica hashing
- Valida entradas erróneas

## ✓ StatsServiceTest

- Cálculo correcto del ratio
- Manejo de casos borde

## ✓ MutantControllerTest

- 200 mutante
- 403 humano
- Validaciones y errores
- Integración básica con MockMvc

---

## 7. Cobertura de Código – Jacoco

mutantes												
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cqty	Missed	Lines	Missed	Methods	Missed	Classes
org.example.mutantes.exception	48 %	n/a	5	9	12	24	5	9	1	2	0	2
org.example.mutantes.service	94 %	92 %	6	53	4	64	0	7	0	3	0	3
org.example.mutantes.validation	88 %	68 %	5	10	4	15	0	2	0	1	0	1
org.example.mutantes	37 %	n/a	1	2	2	3	1	2	0	2	0	1
org.example.mutantes.controller	100 %	100 %	0	3	0	5	0	2	0	1	0	1
org.example.mutantes.entity	100 %	50 %	1	2	0	3	0	1	0	1	0	1
org.example.mutantes.dto	100 %	n/a	0	1	0	1	0	1	0	1	0	1
Total	82 of 673	87 %	13 of 112	88 %	18	80	22	115	6	24	1	10

Resumen del análisis de cobertura:

- **87% de instrucciones cubiertas**

- **88% de branches**
  - Alta cobertura en:
    - MutantDetector
    - MutantService
    - StatsService
    - Controladores
  - Cobertura parcial en GlobalExceptionHandler (normal y aceptable)
- 

## 8. Decisiones Técnicas

### ✓ Hash SHA-256 en vez de guardar ADN crudo

Evita duplicados, reduce memoria y mejora rendimiento.

### ✓ Detector propio $O(N^2)$

Suficientemente eficiente para matrices de cualquier tamaño.

### ✓ Validaciones mínimas y precisas

Solo se validan aspectos necesarios:

- Matriz NxN
- Longitudes iguales
- Caracteres válidos
- No vacío

### ✓ Persistencia en H2 en memoria

Simplifica ejecución y testing.

### ✓ Tests exhaustivos

Alta cobertura, pruebas claras y repetibles.

---

## 9. Conclusiones

- La solución cumple **todos los requisitos del examen**.

- Presenta una **arquitectura limpia y extensible**.
  - Tiene **documentación completa** (Swagger + Informe).
  - Alto nivel de **cobertura de código**.
  - Lista para ser evaluada y presentada.
- 

## 10. Datos del Autor

**Juan Gallardo**

Estudiante de Ingeniería en Sistemas – UTN FRM

GitHub: *(tu enlace al repo)*