

# GLOBAL: DESARROLLO DE SOFTWARE

## 1. Información del Proyecto

- Título: Examen Mercado Libre
- Autor: Oscar Juan Pablo Ortega Rivero
- Fecha: 26/11/2025
- Repositorio GitHub: <https://github.com/juampior/MutantesML-Global3k9>
- URL de la API (Render): <https://mutantesml-global3k9.onrender.com/stats>

## 2. Introducción

El objetivo de este proyecto es desarrollar una API REST eficiente capaz de detectar si un humano es mutante basándose en su secuencia de ADN. El sistema recibe una matriz de secuencias (NxN), la analiza buscando patrones específicos y almacena los resultados para generar estadísticas de verificaciones.

El proyecto ha sido desarrollado utilizando Java 17 y Spring Boot 3, desplegado en la nube mediante Render y contenedorizado con Docker.

## 3. Arquitectura y Diseño

El sistema sigue una Arquitectura en Capas (Layered Architecture) para asegurar la separación de responsabilidades, mantenibilidad y escalabilidad.

### 3.1. Controlador (MutanteController)

Es el punto de entrada de la API. Se encarga de recibir las peticiones HTTP, validar el formato de entrada mediante anotaciones (`@Valid`, `@NotNull`, `@ValidDnaSequence`) y delegar la lógica de negocio a los servicios.

- Implementa documentación con OpenAPI/Swagger (`@Operation`, `@ApiResponse`) para facilitar el consumo de la API.
- Maneja los códigos de respuesta HTTP según el requerimiento: 200 OK para mutantes y 403 Forbidden para humanos.

### 3.2. Servicio (MutanteService)

Contiene la lógica core del negocio. Implementa una estrategia de optimización basada en Hashing y Caché para evitar re-procesar ADN's ya verificados:

1. Generación de Hash: Se calcula un hash único SHA-256 de la secuencia de ADN recibida.
2. Verificación en Base de Datos: Antes de ejecutar el algoritmo de búsqueda de

patrones, se consulta el repositorio buscando si ese hash ya existe.

3. Análisis: Si el ADN es nuevo, se delega al MutanteDetector para buscar las secuencias.

4. Persistencia: El resultado se guarda en la base de datos H2 asociado al hash, asegurando que solo haya 1 registro por ADN.

### 3.3. DTOs y Validación

Se utilizan objetos de transferencia de datos (DnaRequest) para desacoplar la API de la entidad de base de datos. Se implementaron validaciones personalizadas (@ValidDnaSequence) para asegurar que la matriz sea cuadrada y contenga solo caracteres válidos (A, T, C, G) antes de iniciar cualquier procesamiento.

## 4. Algoritmo de Detección

La detección de mutantes se realiza recorriendo la matriz de ADN (Array de Strings) en busca de más de una secuencia de cuatro letras iguales.

Estrategia de Búsqueda:

El algoritmo recorre la matriz en tres direcciones:

1. Horizontal: Filas de izquierda a derecha.
2. Vertical: Columnas de arriba a abajo.
3. Oblicua: Diagonales principales y secundarias.

Optimizaciones:

- Terminación Anticipada (Early Exit): El algoritmo mantiene un contador de secuencias encontradas. En el instante en que el contador es mayor a 1, el proceso se detiene y retorna true, evitando iteraciones innecesarias en el resto de la matriz.
- Validación de Estructura: Se valida la integridad de la matriz (NxN) antes de la ejecución para evitar errores de índice (IndexOutOfBoundsException).

## 5. Diagrama de Secuencia

El código fuente de los diagramas de secuencia se encuentran disponibles en el repositorio del proyecto bajo el nombre: DiagramaSecuenciaPost.puml y DiagramaSecuenciaGet.puml respectivamente.

## 6. Coverage de Tests

Se realizaron pruebas unitarias y de integración para asegurar la calidad del código, alcanzando un coverage superior al 80%, cumpliendo con los requisitos de calidad del Nivel

Element ^	Class, %	Method, %	Line, %	Branch, %
com.example.MutantesML_Global3k9	92% (12/13)	87% (51/58)	84% (201/239)	75% (123/164)
config	100% (1/1)	100% (1/1)	100% (18/18)	100% (0/0)
SwaggerConfig	100% (1/1)	100% (1/1)	100% (18/18)	100% (0/0)
controller	100% (1/1)	100% (4/4)	100% (14/14)	100% (2/2)
MutantController	100% (1/1)	100% (4/4)	100% (14/14)	100% (2/2)
dto	100% (3/3)	100% (18/18)	100% (18/18)	46% (28/60)
DnaRequest	100% (1/1)	100% (4/4)	100% (4/4)	50% (4/8)
ErrorResponse	100% (1/1)	100% (8/8)	100% (8/8)	42% (17/40)
StatsResponse	100% (1/1)	100% (6/6)	100% (6/6)	58% (7/12)
entity	100% (1/1)	57% (4/7)	72% (8/11)	100% (0/0)
DnaRecord	100% (1/1)	57% (4/7)	72% (8/11)	100% (0/0)
exception	100% (2/2)	57% (4/7)	54% (24/44)	100% (0/0)
DnaHashCalculationException	100% (1/1)	50% (1/2)	50% (2/4)	100% (0/0)
GlobalExceptionHandler	100% (1/1)	60% (3/5)	55% (22/40)	100% (0/0)
repository	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
DnaRecordRepository	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
service	100% (3/3)	100% (16/16)	92% (98/106)	92% (81/88)
MutantDetector	100% (1/1)	100% (8/8)	92% (62/67)	91% (71/78)
MutantService	100% (1/1)	100% (4/4)	89% (25/28)	100% (6/6)
StatsService	100% (1/1)	100% (4/4)	100% (11/11)	100% (4/4)
validation	100% (1/1)	100% (4/4)	80% (21/26)	85% (12/14)
ValidDnaSequence	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
ValidDnaSequenceValidator	100% (1/1)	100% (4/4)	80% (21/26)	85% (12/14)
MutantesMLGlobal3k9Application	0% (0/1)	0% (0/1)	0% (0/2)	100% (0/0)

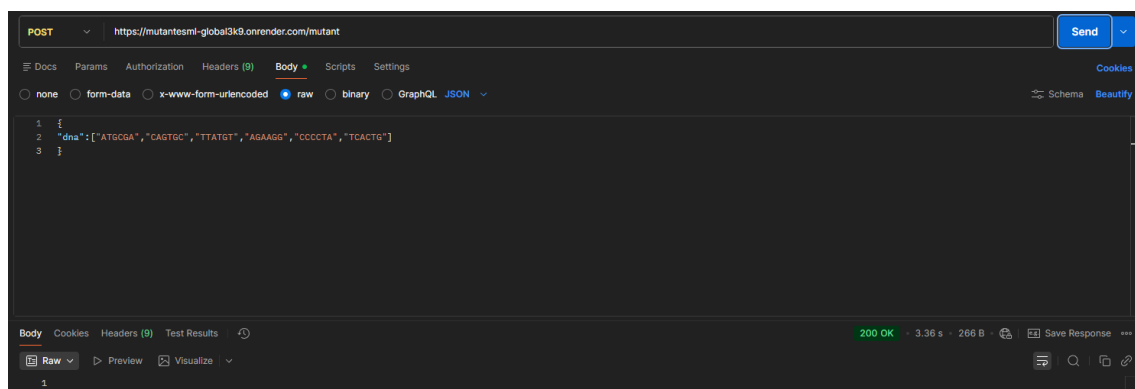
## 7. Pruebas de Funcionamiento con Postman (Evidencia)

A continuación se presentan las pruebas de consumo de la API desplegada en Render mediante Postman.

### 7.1. Detección de Mutante (Caso Positivo)

Se envía una secuencia de ADN mutante. La API procesa la matriz, detecta más de una secuencia de 4 letras y responde con HTTP 200 OK.

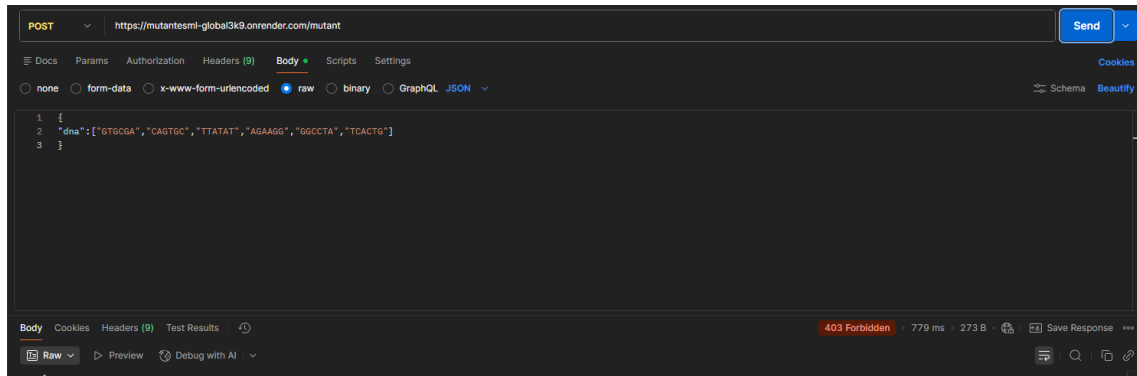
```
{
  "dna":["ATGCGA","CAGTGC","TTATGT","AGAAGG","CCCCTA","TCACTG"]
}
```



### 7.2. Detección de Humano (Caso Negativo)

Se envía una secuencia de ADN sin patrones mutantes. La API responde con HTTP 403 Forbidden.

```
{  
  "dna":["GTGCGA","CAGTGC","TTATAT","AGAAGG","GGCCTA","TCACTG"]  
}
```



### 7.3. Consulta de Estadísticas

Se consulta el endpoint `/stats`. La API devuelve el conteo actualizado de verificaciones y el ratio calculado desde la base de datos.

