Universidad Tecnológica Nacional

Facultad Regional Mendoza

Examen MercadoLibre

Proyecto de Desarrollo de Software - Detector de Mutantes

Alumno: Adriel Espejo
Legajo: 47664
Comisión: 3K9
Materia: Desarrollo de Software
Año: 2025

Documentación Técnica y Entrega Final

Índice

1. Introducción

Magneto quiere reclutar la mayor cantidad de mutantes para poder luchar contra los X-Men. Para ello, ha contratado el desarrollo de un proyecto que detecte si un humano es mutante basándose en su secuencia de ADN.

El objetivo principal es crear un programa con un método o función con la siguiente firma:

```
boolean isMutant(String[] dna);
```

En donde se recibirá como parámetro un array de Strings que representan cada fila de una tabla de (NxN) con la secuencia del ADN. Las letras de los Strings solo pueden ser: (A, T, C, G), las cuales representan cada base nitrogenada del ADN.

2. Desafíos y Niveles Implementados

2.1. Nivel 1: Algoritmo de Detección

Programa en Spring Boot que cumple con el método pedido. Se determina si un humano es mutante si se encuentran **más de una secuencia de cuatro letras iguales**, de forma oblicua, horizontal o vertical.

Ejemplo (Caso mutante):

```
String[] dna = {"ATGCGA", "CAGTGC", "TTATGT", "AGAAGG", "CCCCTA", "
    TCACTG"};
// En este caso el llamado a la función isMutant(dna) devuelve "true".
```

2.2. Nivel 2: API REST

Se creó una API REST hosteada en un cloud computing libre (Render) con el servicio `/mutant/`.

- **Método:** POST
- **Entrada:** JSON con el array de ADN.
- **Salida:**
  - HTTP 200 OK: Si es mutante.
  - HTTP 403 Forbidden: Si no es mutante.

2.3. Nivel 3: Persistencia y Estadísticas

- Se anexó una base de datos **H2** (en memoria) para guardar los ADNs verificados (solo 1 registro por ADN).
- Se expuso el servicio extra `/stats` (GET) que devuelve un JSON con las estadísticas de las verificaciones:

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

```

- Se incluyeron Tests Automáticos con Code Coverage >80 %.

3. Links del Proyecto

A continuación se presentan los enlaces requeridos para la entrega y evaluación del proyecto:

- **Repositorio GitHub (Código Fuente):**
  https://github.com/VerseV/Global-3K9-Adriel-Espejo-47664
- **API en Producción (Render):**
  https://examenmercado-3k9.onrender.com
- **Documentación Swagger UI (Interactiva):**
  https://examenmercado-3k9.onrender.com/swagger-ui.html

4. Arquitectura y Diseño

El proyecto sigue una arquitectura en capas para asegurar escalabilidad, mantenibilidad y testabilidad.

4.1. Componentes Principales

- **Controller:** Maneja las peticiones HTTP (REST).
- **Service:** Contiene la lógica de negocio, el algoritmo de detección y la gestión de caché.
- **Repository:** Interfaz de acceso a datos (JPA) hacia la base de datos H2.
- **Entity:** Representación del modelo de datos (tabla `dna_records`).

4.2. Diagrama de Secuencia (Nivel 3)

A continuación se presenta el diagrama de secuencia que ilustra el flujo completo de una petición de detección de mutantes, incluyendo la validación, verificación de caché, ejecución del algoritmo y persistencia.

[Aquí se insertaría la imagen del Diagrama de Secuencia]
(Ver archivo `docs/Diagrama_de_Secuencia.png` en el repositorio)

Figura 1: Diagrama de Secuencia - Flujo POST /mutant

Nota: El diagrama de secuencia detallado generado con PlantUML se encuentra disponible en la carpeta de documentación del repositorio.

5. Detalles de Implementación y Optimizaciones

5.1. Algoritmo isMutant

Se desarrolló un algoritmo eficiente con las siguientes características:

- **Validación Temprana:** Se verifican dimensiones y caracteres válidos antes de procesar.
- **Conversión Eficiente:** Se transforma el array de Strings a `char[][]` una sola vez para acceso rápido O(1).
- **Early Termination:** El algoritmo se detiene inmediatamente al encontrar más de una secuencia, evitando recorrer el resto de la matriz innecesariamente.

5.2. Estrategia de Caché y Deduplicación

Para cumplir con el requisito de "Solo 1 registro por ADNz optimizar el rendimiento:

1. Se calcula un **Hash SHA-256** único para cada secuencia de ADN recibida.
2. Antes de ejecutar el algoritmo, se consulta la base de datos buscando este Hash.
3. Si existe, se devuelve el resultado almacenado (Caché Hit), ahorrando procesamiento.
4. Si no existe, se procesa, se guarda el resultado y el hash (Caché Miss).

6. Resultados de Testing

Se implementó una suite de pruebas exhaustiva utilizando JUnit 5, Mockito y JaCoCo.

- **Tests Totales:** 37
- **Cobertura de Código:** >90 % (Superando el 80 % requerido)
- **Tipos de Tests:**
  - Tests Unitarios para el algoritmo (casos borde, matrices grandes, etc.).
  - Tests de Integración para los controladores REST.
  - Tests de Servicios con Mocking de repositorios.