Examen Mercadolibre

Magneto quiere reclutar la mayor cantidad de mutantes para poder luchar contra los X-Mens.

Te ha contratado a ti para que desarrolles un proyecto que detecte si un humano es mutante basándose en su secuencia de ADN.

Para eso te ha pedido crear un programa con un método o función con la siguiente firma:

boolean isMutant(String[] dna);

En donde recibirás 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 representa cada base nitrogenada del ADN.

A	T	G	C	G	A
C	A	G	T	G	C
T	T	A	T	T	T
A	G	A	C	G	G
G	C	G	T	C	A
T	C	A	C	T	G

No-Mutante

A	T	G	C	**G**	A
C	**A**	G	T	**G**	C
T	T	**A**	T	**G**	T
A	G	A	**A**	**G**	G
C	**C**	**C**	**C**	T	A
T	C	A	C	T	G

Mutante

Sabrás si un humano es mutante, si encuentras **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”.

Desarrolla el algoritmo de la manera más eficiente posible.

Desafíos:

Nivel 1:
Programa (en spring boot) que cumpla con el método pedido por Magneto.

Nivel 2:
Crear una API REST, hostear esa API en un cloud computing libre (Render), crear el servicio "/mutant/" en donde se pueda detectar si un humano es mutante enviando la secuencia de ADN mediante un HTTP POST con un Json el cual tenga el siguiente formato:

POST → /mutant/
{
"dna":["ATGCGA","CAGTGC","TTATGT","AGAAGG","CCCCTA","TCACTG"]
}

En caso de verificar un mutante, debería devolver un HTTP 200-OK, en caso contrario un 403-Forbidden

Nivel 3:
Anexar una base de datos H2, la cual guarde los ADN's verificados con la API. Solo 1 registro por ADN.
Exponer un servicio extra "/stats" que devuelva un Json con las estadísticas de las verificaciones de ADN: {"count_mutant_dna":40, "count_human_dna":100: "ratio":0.4}

Test-Automáticos, Code coverage > 80%, Diagrama de Secuencia /

Entregar:

- Código Fuente En repositorio github).
- Instrucciones de cómo ejecutar el programa o la API. (En README de github).
- URL de la API ().
- Formato PDF para documentos (Nivel 3).

Análisis Detallado de los Requerimientos
El proyecto presenta un desafío de programación incremental, donde cada nivel construye sobre el anterior, requiriendo un buen diseño y comprensión de conceptos clave.

- Nivel 1: Implementación de la función isMutant
    - Objetivo: Desarrollar un algoritmo que determine si un humano es mutante, basándose en la presencia de secuencias de cuatro letras iguales en su ADN.
    - Entrada: Un array de strings que representa una matriz cuadrada NxN.
    - Salida: Un valor booleano (true si es mutante, false si no lo es).
    - Reglas:
        - Secuencias deben buscarse en horizontal, vertical y oblicua (diagonales).

        - Un humano es mutante si se encuentran *más de una* secuencia de cuatro letras iguales.
        - La función debe ser eficiente.
        -
- **Nivel 2: Creación de una API REST**
    - Objetivo: Exponer la funcionalidad del Nivel 1 a través de una API que pueda ser invocada externamente.
    - Tecnología: Se debe usar Spring Boot y un servicio de cloud computing libre (Render en este caso).
    - Endpoint /mutant/:
        - Método: POST
        - Entrada: JSON con el array de ADN.
        - Salida:
            - HTTP 200 OK si es mutante.
            - HTTP 403 Forbidden si no es mutante.
            -
- **Nivel 3: Persistencia y Estadísticas**
    - Objetivo: Almacenar la información de cada secuencia de ADN analizada y proporcionar estadísticas sobre las verificaciones.
    - Base de Datos: Utilizar H2 (base de datos embebida)
        - Almacenar el ADN verificado.
        - Solo un registro por ADN (evitar duplicados).
    - Endpoint /stats:
        - Método: GET
        - Salida: JSON con las siguientes estadísticas:
            - count_mutant_dna: Número de ADN mutantes verificados.
            - count_human_dna: Número de ADN humanos verificados.
            - ratio: count_mutant_dna / count_human_dna.
    - Consideraciones de tráfico y escalabilidad: Aunque el alcance de este proyecto es limitado, la alta concurrencia potencial requiere considerar aspectos de eficiencia y manejo de recursos.
    -

Guía Paso a Paso para la Resolución

A continuación, te presento una guía detallada para abordar cada nivel del proyecto, incluyendo consejos para la implementación:

Función isMutant(String[] dna)

1. Validación del Input:
    - Verificar que el array dna no sea nulo o vacío.
    - Asegurar que todas las cadenas en el array tengan la misma longitud.
    - Verificar que la longitud del array sea igual a la longitud de cada cadena (matriz cuadrada NxN).
    - Validar que cada caracter sea A, T, C, o G. Si alguna de estas condiciones no se cumple, lanzar una excepción apropiada o devolver false.
    -
2. Estructura del Algoritmo:
    - Iterar sobre la matriz para buscar secuencias de cuatro letras iguales en cada dirección:

        - Horizontal: Verificar cada fila.
        - Vertical: Verificar cada columna.
        - Oblicua (Diagonales): Verificar ambas diagonales principales y todas las diagonales paralelas a ellas.
    - Llevar un contador de secuencias encontradas.
    - Si el contador supera 1, devolver true (es mutante).
    - Si al finalizar las iteraciones el contador es menor o igual a 1, devolver false (no es mutante).
3. Optimización del Algoritmo:
    - Evitar Revisitar: Una vez que se encuentra una secuencia en una dirección, no es necesario seguir buscando en esa fila/columna/diagonal.
    - Terminación Anticipada: Tan pronto como el contador de secuencias sea mayor que 1, terminar la búsqueda y devolver true.
    - Paralelización (Opcional): Se pueden usar hilos o programación paralela para acelerar la búsqueda en diferentes direcciones simultáneamente (especialmente útil para matrices grandes).
    -

Explicación Detallada (Capas del Backend)

- MutantDetectorApplication: Clase principal que inicia la aplicación Spring Boot.
- DnaRecord: Entidad JPA que mapea a la tabla para almacenar la información del ADN (hash y si es mutante o no).
- DnaRecordRepository: Interfaz que extiende JpaRepository para facilitar el acceso a la base de datos para las operaciones relacionadas con la entidad DnaRecord.
- MutantService:
    - Contiene la lógica de negocio para la detección de mutantes (invocando a MutantDetector) y para guardar el resultado en la base de datos.
    - Es responsable de calcular el hash del ADN para evitar duplicados en la base de datos.
- StatsService:
    - Responsable de obtener las estadísticas desde la base de datos, como el número de ADN mutantes, el número de ADN humanos y la proporción entre ambos.
- MutantController:
    - Expone los endpoints REST /mutant (para la detección de mutantes) y /stats (para obtener las estadísticas).
    - Delega las llamadas al servicio correspondiente (MutantService o StatsService) para realizar la lógica de negocio.

Esta arquitectura en capas facilita la testabilidad, el mantenimiento y la escalabilidad de tu proyecto.

Implementación en Java (Ejemplo):

Nivel 3: Persistencia de Datos y Estadísticas

1. Añadir Dependencias: Agregar la dependencia de Spring Data JPA y H2 en pom.xml.
2. Crear Entidad DnaRecord: (aquí iría el código de la entidad)
3. Modificar la Configuración de Spring Boot: Agregar la configuración para usar H2 en el archivo application.properties (o application.yml):

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
```

spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true # Opcional, para ver la consola de H2
spring.jpa.hibernate.ddl-auto=update #Crear la base de datos y actualizarla

4. Despliegue en Render: Realizar nuevamente el despliegue en Render con los cambios.
5. Consideraciones Clave:
    - Test:
        - Escribir tests unitarios para isMutant.
        - Escribir tests de integración para los endpoints de la API.
        - Asegurar un code coverage superior al 80%.

Análisis Detallado de los Requerimientos

El proyecto presenta un desafío de programación incremental, donde cada nivel construye sobre el anterior, requiriendo un buen diseño y comprensión de conceptos clave.

- **Nivel 1: Implementación de la función isMutant**
    - **Objetivo:** Desarrollar un algoritmo que determine si un humano es mutante, basándose en la presencia de secuencias de cuatro letras iguales en su ADN.
    - **Entrada:** Un array de strings que representa una matriz cuadrada NxN.
    - **Salida:** Un valor booleano (true si es mutante, false si no lo es).
    - **Reglas:**
        - Secuencias deben buscarse en horizontal, vertical y oblicua (diagonales).
        - Un humano es mutante si se encuentran *más de una* secuencia de cuatro letras iguales.
        - La función debe ser eficiente.
- **Nivel 2: Creación de una API REST**
    - **Objetivo:** Exponer la funcionalidad del Nivel 1 a través de una API que pueda ser invocada externamente.
    - **Tecnología:** Se debe usar Spring Boot y un servicio de cloud computing libre (Render en este caso).
    - **Endpoint /mutant/:**
        - Método: POST
        - Entrada: JSON con el array de ADN.
        - Salida:
            - HTTP 200 OK si es mutante.
            - HTTP 403 Forbidden si no es mutante.
- **Nivel 3: Persistencia y Estadísticas**
    - **Objetivo:** Almacenar la información de cada secuencia de ADN analizada y proporcionar estadísticas sobre las verificaciones.
    - **Base de Datos:** Utilizar H2 (base de datos embebida)
        - Almacenar el ADN verificado.
        - Solo un registro por ADN (evitar duplicados).
    - **Endpoint /stats:**
        - Método: GET
        - Salida: JSON con las siguientes estadísticas:
            - count_mutant_dna: Número de ADN mutantes verificados.

        - count_human_dna: Número de ADN humanos verificados.
        - ratio: count_mutant_dna / count_human_dna.
    - Consideraciones de tráfico y escalabilidad: Aunque el alcance de este proyecto es limitado, la alta concurrencia potencial requiere considerar aspectos de eficiencia y manejo de recursos.

Guía Paso a Paso para la Resolución

A continuación, te presento una guía detallada para abordar cada nivel del proyecto, incluyendo consejos para la implementación:

Función isMutant(String[] dna)

1. **Validación del Input:**
    - Verificar que el array dna no sea nulo o vacío.
    - Asegurar que todas las cadenas en el array tengan la misma longitud.
    - Verificar que la longitud del array sea igual a la longitud de cada cadena (matriz cuadrada NxN).
    - Validar que cada caracter sea A, T, C, o G. Si alguna de estas condiciones no se cumple, lanzar una excepción apropiada o devolver false.
2. **Estructura del Algoritmo:**
    - Iterar sobre la matriz para buscar secuencias de cuatro letras iguales en cada dirección:
        - **Horizontal:** Verificar cada fila.
        - **Vertical:** Verificar cada columna.
        - **Oblicua (Diagonales):** Verificar ambas diagonales principales y todas las diagonales paralelas a ellas.
    - Llevar un contador de secuencias encontradas.
    - Si el contador supera 1, devolver true (es mutante).
    - Si al finalizar las iteraciones el contador es menor o igual a 1, devolver false (no es mutante).
3. **Optimización del Algoritmo:**
    - **Evitar Revisitar:** Una vez que se encuentra una secuencia en una dirección, no es necesario seguir buscando en esa fila/columna/diagonal.
    - **Terminación Anticipada:** Tan pronto como el contador de secuencias sea mayor que 1, terminar la búsqueda y devolver true.
    - **Paralelización (Opcional):** Se pueden usar hilos o programación paralela para acelerar la búsqueda en diferentes direcciones simultáneamente (especialmente útil para matrices grandes).

4. Explicación Detallada

- **MutantDetectorApplication:** Clase principal que inicia la aplicación Spring Boot.
- **DnaRecord:** Entidad JPA que mapea a la tabla para almacenar la información del ADN (hash y si es mutante o no).
- **DnaRecordRepository:** Interfaz que extiende JpaRepository para facilitar el acceso a la base de datos para las operaciones relacionadas con la entidad DnaRecord.
- **MutantService:**
    - Contiene la lógica de negocio para la detección de mutantes (invocando a MutantDetector) y para guardar el resultado en la base de datos.
    - Es responsable de calcular el hash del ADN para evitar duplicados en la base de datos.
- **StatsService:**

- Responsable de obtener las estadísticas desde la base de datos, como el número de ADN mutantes, el número de ADN humanos y la proporción entre ambos.

- **MutantController:**
  - Expone los endpoints REST /mutant (para la detección de mutantes) y /stats (para obtener las estadísticas).
  - Delega las llamadas al servicio correspondiente (MutantService o StatsService) para realizar la lógica de negocio.

  Esta arquitectura en capas facilita la testabilidad, el mantenimiento y la escalabilidad de tu proyecto.

5.Implementación en Java (Ejemplo):

Nivel 3: Persistencia de Datos y Estadísticas

1. **Añadir Dependencias:** Agregar la dependencia de Spring Data JPA y H2 en pom.xml.
2. **Crear Entidad DnaRecord:**
6. **Modificar la Configuración de Spring Boot:** Agregar la configuración para usar H2 en el archivo application.properties (o application.yml):

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true # Opcional, para ver la consola de H2
spring.jpa.hibernate.ddl-auto=update  #Crear la base de datos y actualizarla
```

7. **Despliegue en Render:** Realizar nuevamente el despliegue en Render con los cambios.
   - .

8 . Consideraciones Clave:

- **Test:**
  - Escribir tests unitarios para isMutant.
  - Escribir tests de integración para los endpoints de la API.
  - Asegurar un code coverage superior al 80%.