Informe: Parcial mutantes

Emiliano Jordan 50061

DESARROLLO DE SOFTWARE

CONTENIDO

INTRODUCCIÓN 2
CONSIGNAS DEL DESAFÍO 2
Desafíos: 3
Nivel 1: 3
Nivel 2: 3
Nivel 3: 3
FUNCIONAMIENTO GENERAL DEL PROGRAMA 3
ALGORITMO DE RESOLUCIÓN 4
Is Mutant (metodo principal) 4
Revisar filas 5
Revisar columnas 7
Revisar diagonales 7
Explicación ilustrativa 8
Conclusión del algoritmo 9
PRUEBAS CON POSTMAN 9
Post /mutant/ 9
Petición 1 9
Petición 2 10
GET /STATS/ 10
Request 10
Response 10
DIAGRAMAS DE SECUENCIA 11
Post 11
GET 12
JMeter 13
JUnit – Pruebas Unitarias 13
Resultado de los Test 14
Code coverage 14

INTRODUCCIÓN

El siguiente informe está hecho para detallar información sobre el funcionamiento general del programa, cómo funciona el algoritmo de resolución, mostrar cómo se comporta ante distintos test y pruebas de stress, y cómo funciona el programa explicado mediante diagramas de secuencia UML.

CONSIGNAS DEL DESAFÍO

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 programado en el lenguaje Java con spring boot. Crea las clases necesarias y la verificación en el método main.

DESAFÍOS:

NIVEL 1:

Programa en java spring boot que cumpla con el método pedido por Magneto utilizando una arquitectura en capas de controladores, servicios y repositorios.

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 en 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}

Tener en cuenta que la API puede recibir fluctuaciones agresivas de tráfico (Entre 100 y 1 millón de peticiones por segundo). Utilizar Jmeter

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

FUNCIONAMIENTO GENERAL DEL PROGRAMA

El programa tiene dos funciones principales, verificar si un dna es o no mutante, y la otra dar las estadísticas de los datos analizados hasta el momento.

- Verificar DNA (POST /mutant/): para verificar un dna, el usuario debe ingresar la matriz (arreglo de Strings) por teclado y enviarla como body de la petición. El programa ejecutará una función llamada "isMutant" la cual devolverá **true** solo si encuentra **DOS SECUENCIAS DISTINTAS** de cuatro letras iguales de forma oblicua, horizontal o vertical; si no encuentra dos secuencias distintas, devolverá **false**

- Obtener estadísticas (GET /stats/): cuando el usuario hace el GET al endpoint /stats/, el programa busca en la base de datos la cantidad de dna que le corresponde a mutantes, y la que le corresponde a humanos. Luego le muestra al usuario la cantidad de cada uno, junto con la proporción de ADN mutante en comparación con el ADN humano, llamada "ratio".

ALGORITMO DE RESOLUCIÓN

Se buscó realizar el algoritmo de una manera eficiente, por lo tanto, leer cada elemento de la matriz no era opción. El algoritmo utilizado se basó e inspiró en lo redactado en el siguiente artículo (digo inspiró ya que no funciona exactamente como dice el escritor):
https://medium.com/@elbrunorey/mi-experiencia-con-el-desaf%C3%ADo-de-mercadolibre-3f7b4ec6493

El cual, dicho en pocas palabras, explica que no hace falta leer cada elemento de la matriz para saber si hay una secuencia de 4 letras iguales, por lo que el programa tiene dos partes fundamentales.

- Primero el programa va leyendo los elementos de la matriz de forma intercalada, empezando siempre desde el primer elemento de la fila/columna/diagonal. Si dos elementos, separados por un lugar de la matriz, son iguales, entonces podría haber una secuencia en ese lugar.
- Cada vez que se encuentra una posible secuencia en el paso anterior, se empiezan a ver los elementos que antes fueron salteados para verificar si hay o no una secuencia de letras.

Aclaración: de ahora en más, cuando se menciona la frase "posible secuencia", se refiere a cuando se encuentra que dos elementos, con un espacio en el medio, son iguales.

El algoritmo tiene un método principal llamado "isMutant", el cual recibe como parámetro el dna (arreglo de Strings) y retorna un booleano.

La verificación del dna fue dividido en 7 métodos, los cuales tienen funciones específicas: el primero es para revisar las filas, el segundo para revisar las columnas, el tercer método se encarga de llamar a los siguientes cuatro, los cuales revisan las diagonales tanto derechas como izquierdas. Los tres primeras métodos son llamadas desde un método principal llamado "isMutant" el cual determinará si un dna es o no mutante.

Los 7 métodos comparten los mismos parámetros:

- dna (String []): es la matriz ingresada por el usuario y la que van a evaluar las funciones
- dimension (int): es el length del arreglo, pero restado uno, ya que el arreglo empieza a indexar desde el cero
- secuenciasEncontradas(String): es un String en el cual, una vez que se encuentre una secuencia, vamos a almacenar la letra que le corresponde a la secuencia encontrada. Esto es para que, cuando vayamos a evaluar otras posibles secuencias, no se evalúen secuencias que ya hemos encontrado, ya que justamente debemos encontrar dos secuencias distintas.

Además, estos 7 métodos retornan la variable *secuenciasEncontradas* la cual será verificada en el método "isMutant".

IS MUTANT (METODO PRINCIPAL)

Este método es el primero que se ejecuta. Este método contiene llama a los siguientes métodos:

1. revisarFilas
2. revisarColumnas
3. revisarDiagonales
    a. revisarDiagonalDerechaColumna

b. revisarDiagonalIzquierdaColumna
c. revisarDiagonalDerechaFila
d. revisarDiagonalIzquierdaFila

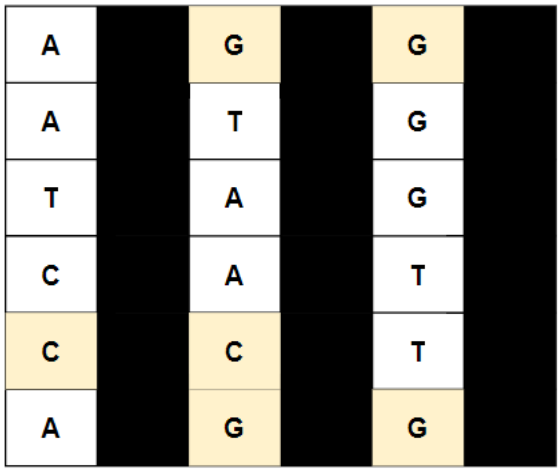
Luego de cada llamada se verifica si *secuenciasEncontradas* contiene dos caracteres, si es así, devuelve **true**; en caso contrario sigue con la ejecución de cada método.

Si luego de la ejecución de "revisarDiagonales" *secuenciasEncontradas* no contiene dos caracteres, devuelve **false.**

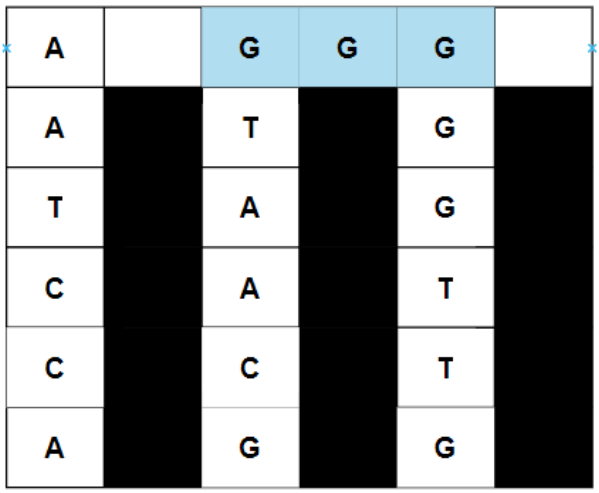
REVISAR FILAS

Los dos pasos explicados anteriormente lo hacen cada método, solo que en la dirección que le corresponde a cada función.

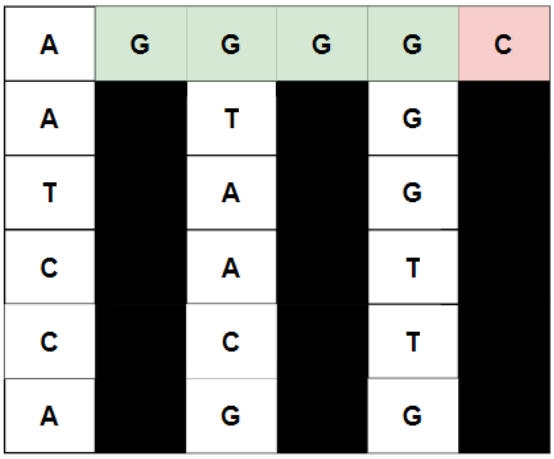
1) Primero se van recorriendo de manera intercalada, partiendo siempre desde el primer elemento de la fila. Cada vez que encuentra una posible secuencia (pintadas de amarillo), se evalúa que la letra no le corresponda a una secuencia ya encontrada (es decir, vemos si la letra está en el String *secuenciasEncontradas*), si no está, se ejecuta el paso 2, sino se siguen evaluando los elementos de la fila.



2) Una vez que se encuentra una posible secuencia, se procede a ver si el elemento que se encuentra en el medio de los dos elementos es igual a ellos, ya que, si el elemento del medio no es igual a los otros dos, no hay forma de generar una secuencia de cuatro letras.
Esto optimiza el tiempo ya que leemos los elementos de los extremos solo cuando realmente puede llegar a haber una secuencia. Si el elemento del medio es igual, se evalúan los del extremo; sino se sigue ejecutando el paso 1 con los demás elementos de la matriz



3) Una vez que se verificó que el elemento del medio es igual, se evalúan los extremos. Está contemplado que cuando el uno de los elementos que forman la posible secuencia (elementos pintados de amarillo en el paso 1) se encuentren en los extremos de la matriz, no habrá elemento izquierdo o derecho, por lo que esto se valida antes de realizar la evaluación de los extremos.
Si alguno de los extremos es igual a los demás elementos, hemos encontrado una secuencia.



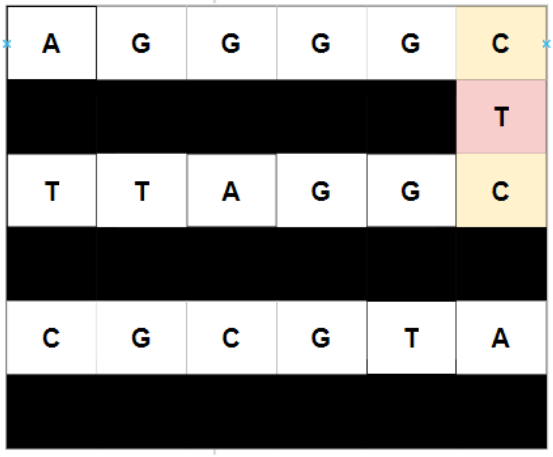
4) Finalmente, luego de encontrar una secuencia, se agrega la letra de dicha secuencia a la variable *secuenciasEncontradas*, para evitar evaluar posibles secuencias repetidas.
Además, cada vez que se encuentra una secuencia, se evalúa si *secuenciasEncontradas* contiene dos caracteres, ya que, si esto es verdad, significa que ya hemos encontrado dos secuencias distintas de cuatro letras, por lo que no importa seguir evaluando la matriz, ya que, haya o no más secuencias, el dna ya le corresponde a un mutante.
Si la validación anterior se cumple, termina la ejecución del método y retorna al método principal "isMutant". Si no se cumple, significa que solo se ha encontrado una secuencia, por lo que continúa ejecutándose el paso 1 desde el elemento donde se quedó por última vez.

REVISAR COLUMNAS

El funcionamiento es exactamente al "Revisar filas", solo que de manera vertical. No se explicarán los pasos ya que son los mismos, pero se agregarán imágenes para ilustrar los pasos.

1) Se encuentras dos posibles secuencias, pero la posible secuencias de "G" es descartada ya que ya existe una secuencia de esa letra.

2) Como el elemento del medio no es igual, ya no existe la posibilidad de que haya una secuencia en ese lugar, por lo que sigue evaluando el paso 1 desde el lugar donde se quedó. Si el elemento del medio hubiera sido igual, hubiera hecho los mismo pasos que el "Revisar Filas" pero con las columnas.



REVISAR DIAGONALES

Esta función en sí no mira la matriz, simplemente llama a las otras 4 funciones restantes, las cuales se encargan de revisar las diagonales derechas e izquierdas, pero ¿por qué 4 funciones para ver las diagonales si solo hay diagonales derechas e izquierdas?

Se optó por dividir la tarea de revisar las diagonales derechas e izquierdas en dos, por lo que hay dos funciones que evalúan las diagonales derechas, y dos funciones que evalúan las diagonales izquierdas.

Esto es, porque, si bien las diagonales derechas son solo un grupo, las diagonales que están a la derecha de la diagonal principal se acceden cambiando la posición de la columnas; mientras que las diagonales derechas que están debajo de la diagonal principal se acceden cambiando la posición de las filas. Pasa exactamente lo mismo con las diagonales izquierdas.

Por ende, el método "revisarDiagonales", llama a los cuatro métodos restantes, las cuales son:

- revisarDiagonalDerechaColumna: evalúa las diagonales derechas, iterando sobre la posición de las columnas
- revisarDiagonalIzquierdaColumna: evalúa las diagonales izquierdas, iterando sobre la posición de las columnas
- revisarDiagonalDerechaFila: evalúa las diagonales derechas, iterando sobre la posición de las filas
- revisarDiagonalIzquierdaFila: evalúa las diagonales izquierdas, iterando sobre la posición de las filas

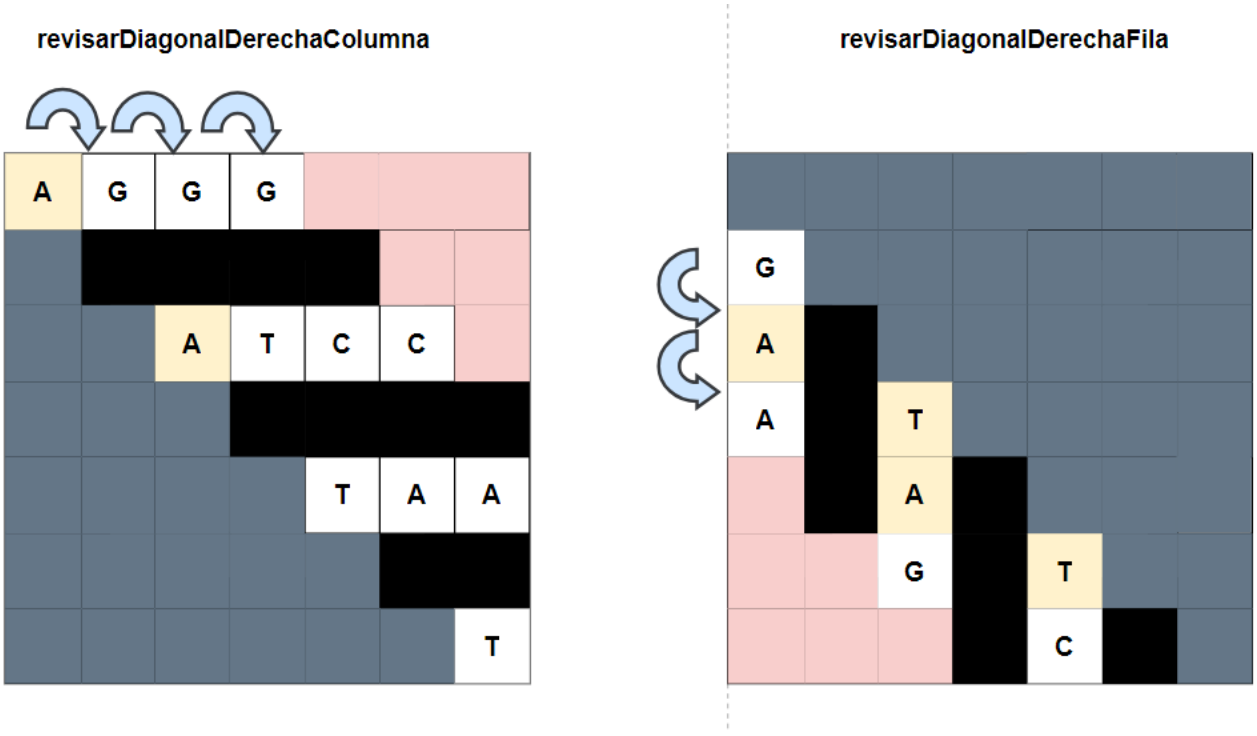
El algoritmo de verificación de cada método funciona con la misma lógica de filas y columnas, solo que aplicado a las diagonales.

Además, el método padre "revisarDiagonales" después de cada ejecución de cada uno de los métodos individuales, revisa si *secuenciasEncontradas* contiene 2 caracteres, para evitar seguir revisando secuencias.

EXPLICACIÓN ILUSTRATIVA

A continuación, se proporciona unas imágenes para entender mejor esto de dividir la búsqueda de diagonales derechas e izquierdas en dos. Las imágenes corresponden a los métodos de evaluar diagonales derechas, pero es análogo para las diagonales izquierdas.

Como se puede ver en la imagen, si bien las dos imágenes contienen diagonales derechas, para poder acceder a las diagonales de la primer imagen hay que ir cambiando de columnas; mientras que para acceder a las diagonales de la segunda imagen se debe ir cambiando de fila. Por este motivo se decidió separar la tarea de revisar diagonales en 4 métodos (2 para las derechas y 2 para las izquierdas).



En ambas imágenes se pueden apreciar celdas en color rojo, esto es porque, en esos lugares es imposible que haya una secuencia diagonal de 4 elementos, ya que no hay celdas suficientes para lograr una secuencia.

Además, cuando se iteran las filas (método "revisarDiagonalDerechaFila") se puede apreciar que no toma en cuenta la diagonal principal, ya que esta ya fue evaluada en la método "revisarDiagonalDerechaColumna".

CONCLUSIÓN DEL ALGORITMO

Se pudo observar que, al realizar este algoritmo solo vemos la mitad de los elementos de la matriz. Además, al verificar si ya hay dos secuencias encontradas después de encontrar una de ellas, evita seguir revisando la matriz, por lo que optimizamos muchísimo el tiempo de verificación de un dna.

PRUEBAS CON POSTMAN

El proyecto fue deployado en Render el cual se puede acceder desde el siguiente link:

https://parcial-mutantes-qs03.onrender.com/

A continuación se dejarán algunas imágenes para mostrar las pruebas que fueron ejecutadas desde Postman para los métodos POST y GET.

POST /MUTANT/

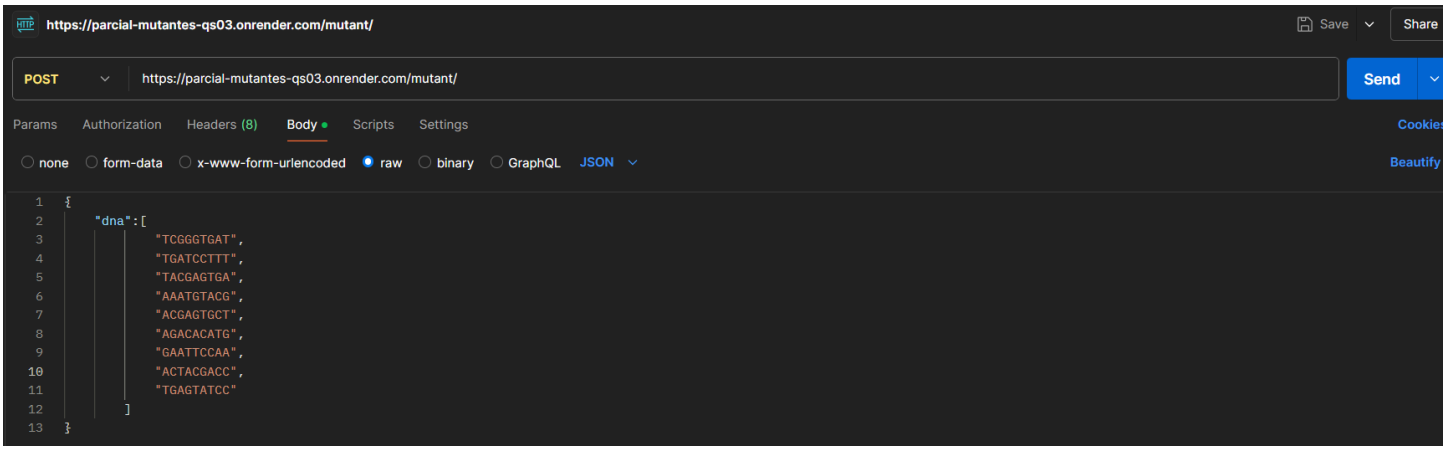
Para realizar el post se debe apuntar al endpoint /mutant/. Además, el body de la petición debe contener el dna (arreglo de Strings), el cual representa el dna a evaluar.

Se adjuntarán capturas de 2 peticiones:

- La primera será de un dna que cumple con las condiciones para ser un mutante. Por lo que se espera un **true** en la respuesta y el estado debe ser un **200 OK**
- La segunda será de un dna que le corresponde a un humano. Por lo que debe devolver **false** en la respuesta y el estado será 403 Forbidden.

PETICIÓN 1

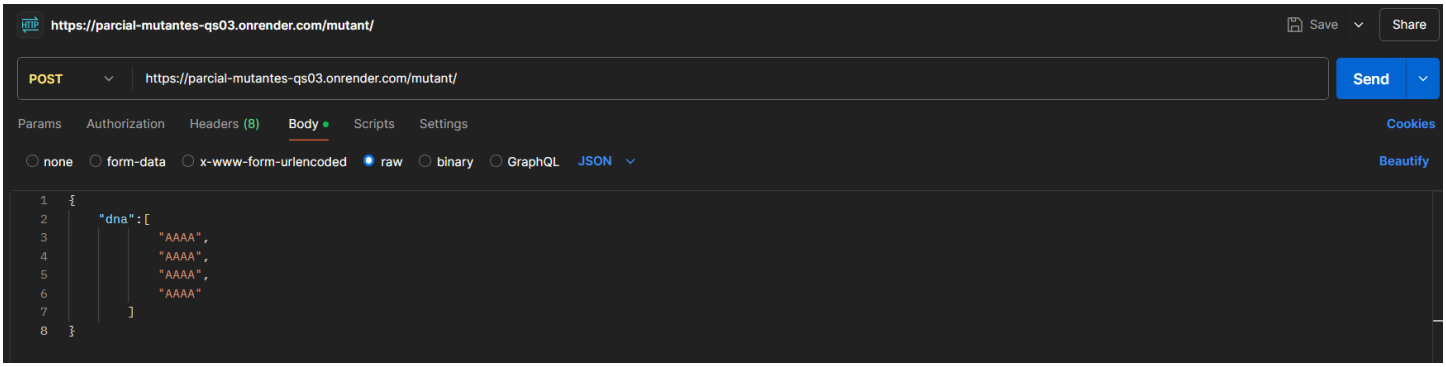
REQUEST



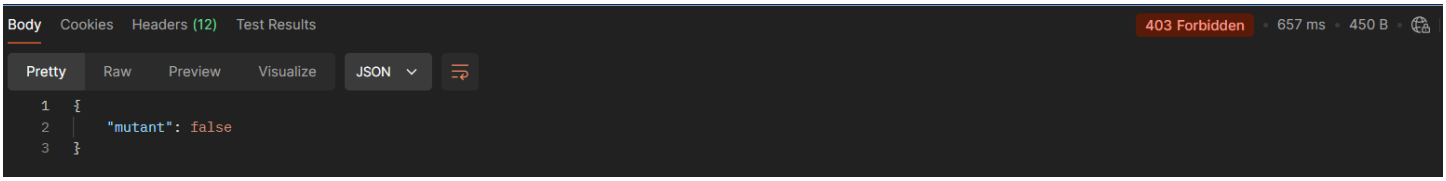
RESPONSE

PETICIÓN 2

REQUEST



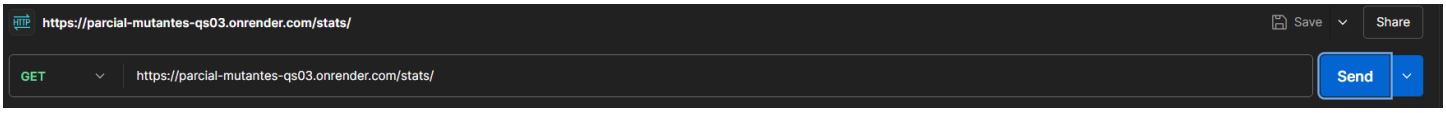
RESPONSE



GET /STATS/

Para realizar el GET se debe apuntar el endpoint /stats/. Al realizar la petición devolverá un JSON con la cantidad de mutantes y humanos verificados, junto con la proporción de ADN mutante en comparación con el ADN humano.

REQUEST



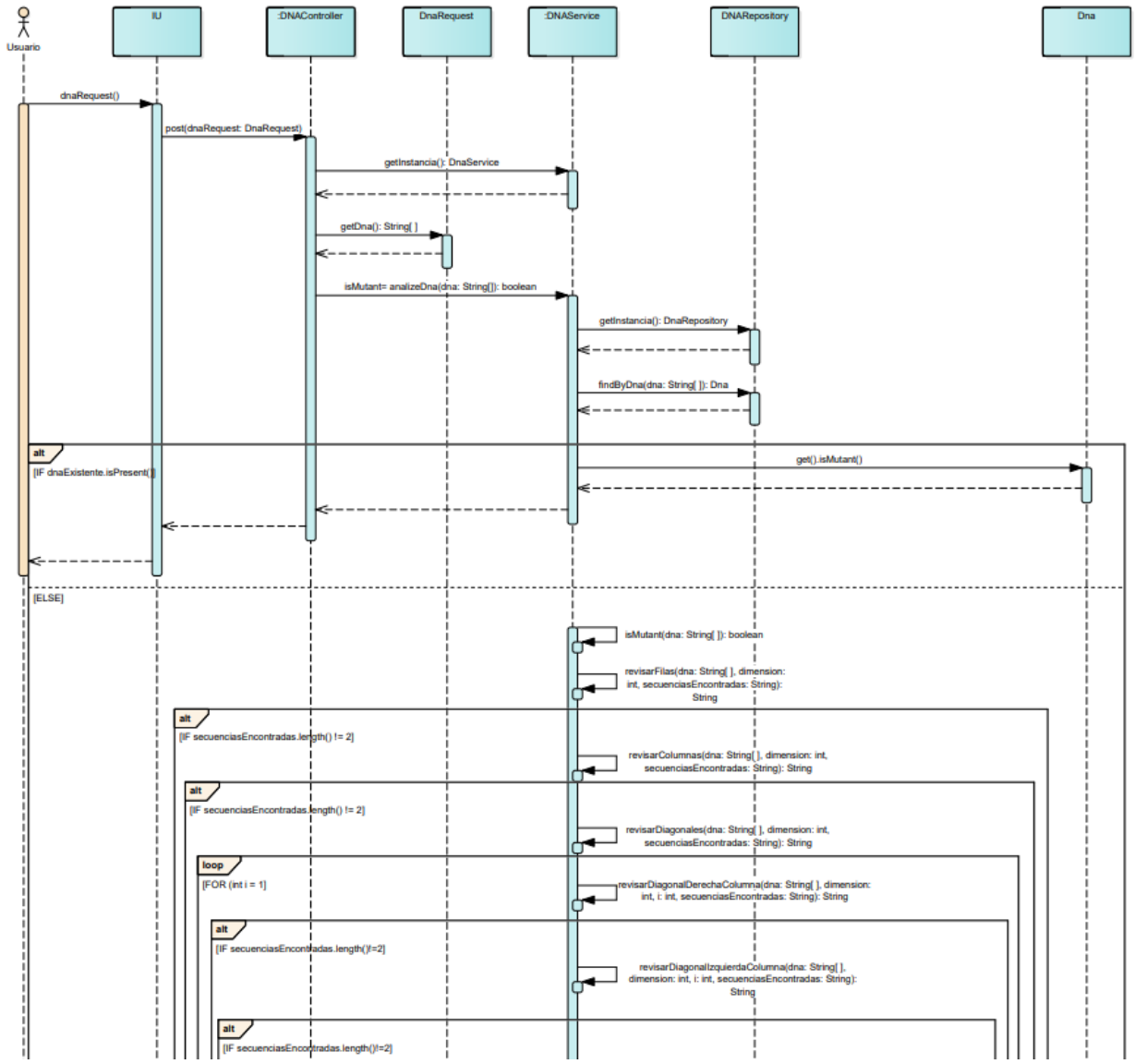
RESPONSE

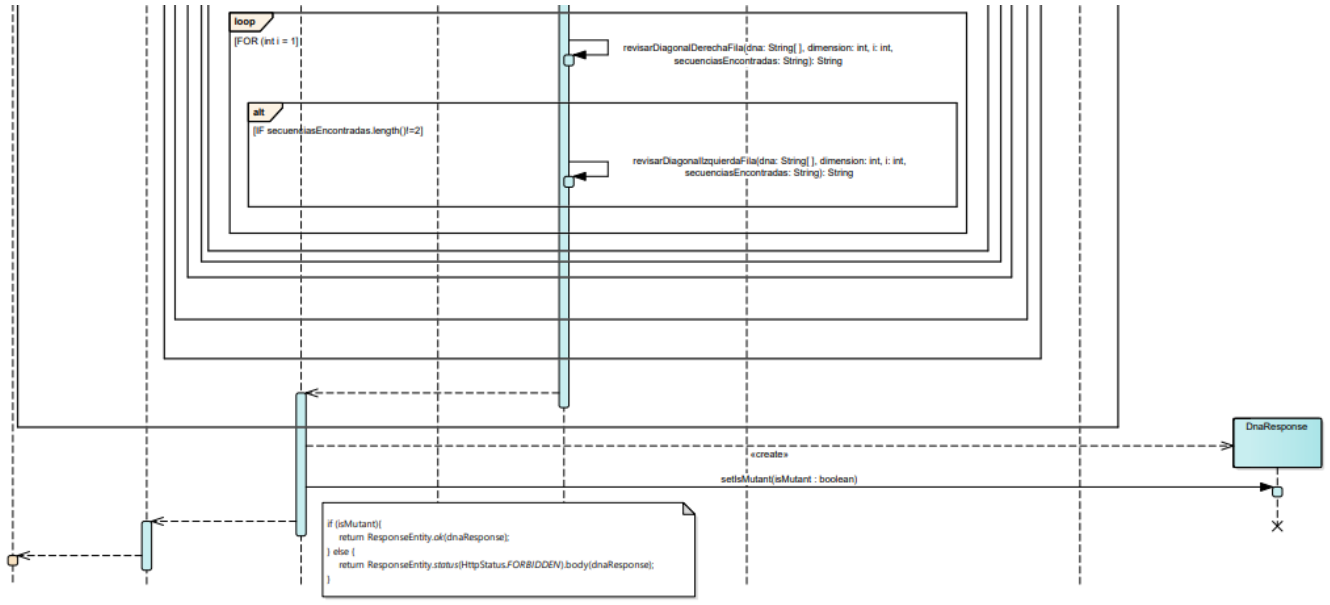
DIAGRAMAS DE SECUENCIA

Se realizaron diagramas de secuencia para explicar a través de un diagrama UML como funcionan las dos peticiones a este proyecto: POST y GET. Se dejarán algunas imágenes para una vista rápida, pero además se adjuntará un link a Drive para poder verlos mucho mejor en formato PDF.

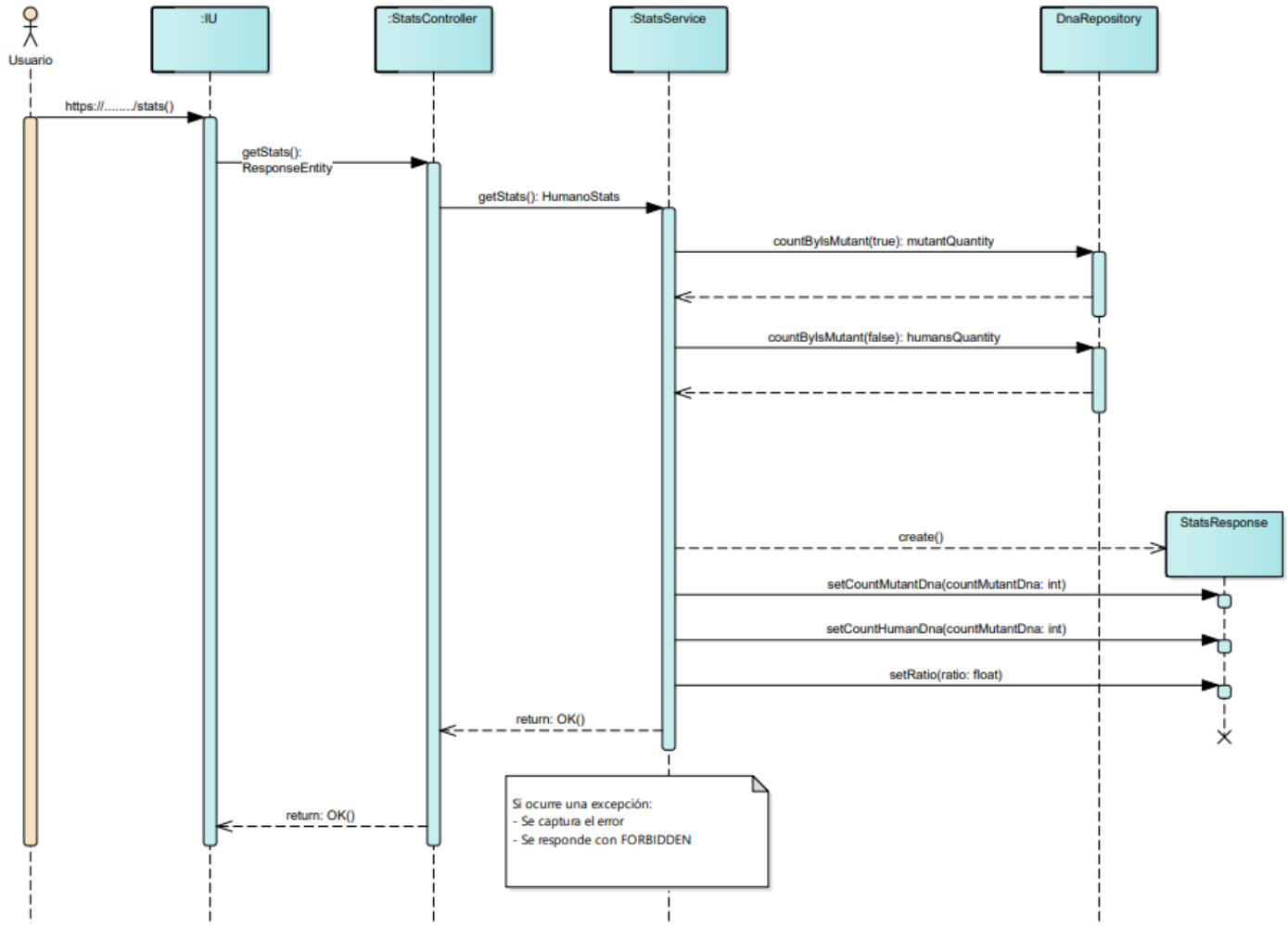
Link al Drive: https://drive.google.com/drive/folders/1v6wT1qfOaiw9_R3c2MRpFLxZVLRc4Tfg?usp=sharing

POST




loop
[FOR (int i = 1)
revisarDiagonalDerechaFila(dna: String[ ], dimension: int, i: int, secuenciasEncontradas: String): String
alt
[IF secuenciasEncontradas.length()!=2]
revisarDiagonalIzquierdaFila(dna: String[ ], dimension: int, i: int, secuenciasEncontradas: String): String
DnaResponse
«create»
setIsMutant(isMutant : boolean)
if (isMutant){
return ResponseEntity.ok(dnaResponse);
} else {
return ResponseEntity.status(HttpStatus.FORBIDDEN).body(dnaResponse);
}

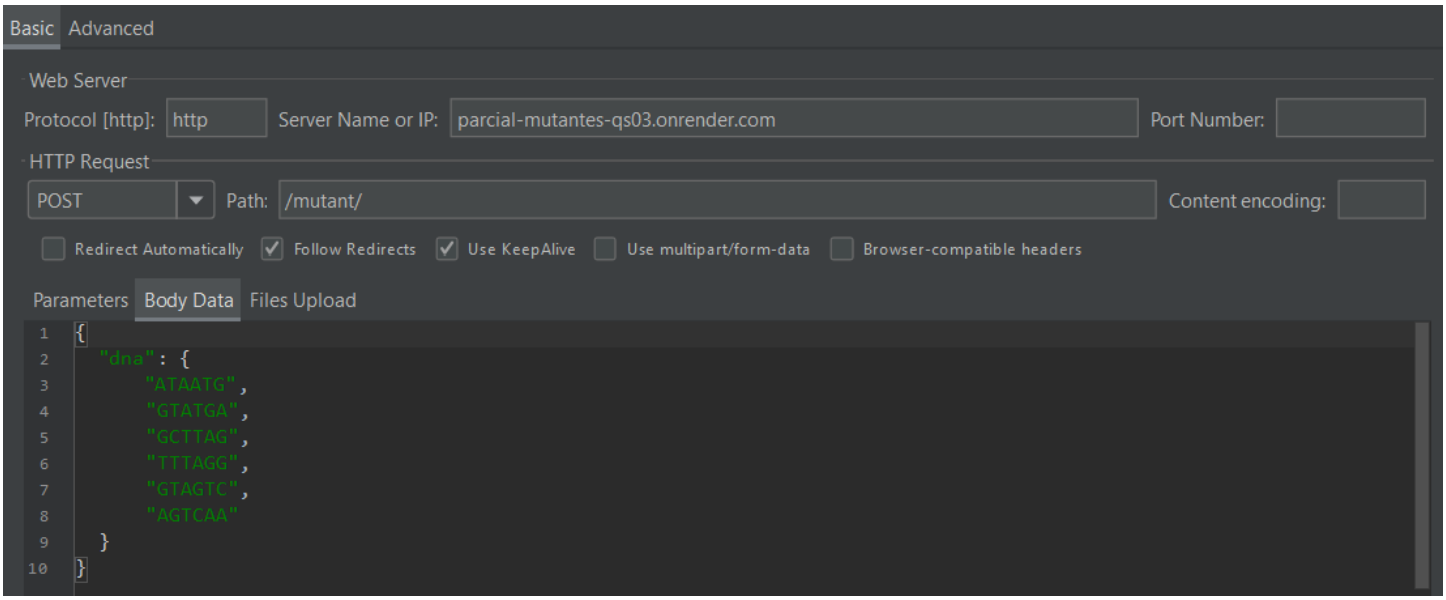
GET

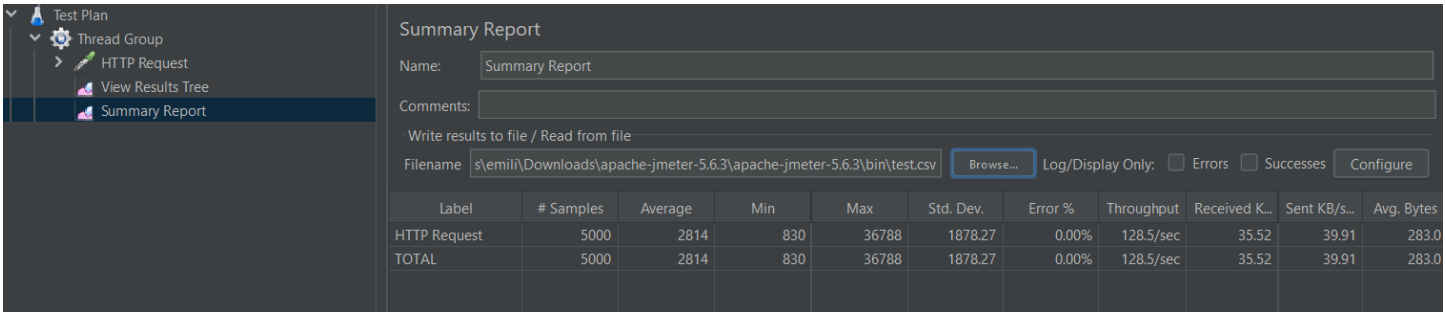

Usuario
:IU
:StatsController
:StatsService
DnaRepository
https://...../stats()
getStats(): ResponseEntity
getStats(): HumanoStats
countByIsMutant(true): mutantQuantity
countByIsMutant(false): humansQuantity
StatsResponse
create()
setCountMutantDna(countMutantDna: int)
setCountHumanDna(countMutantDna: int)
setRatio(ratio: float)
return: OK()
return: OK()
Si ocurre una excepción:
- Se captura el error
- Se responde con FORBIDDEN

JMETER

Se realizaron pruebas de stress al proyecto mediante JMeter.

Aquí hay algunas capturas del resultado de realizarle 5000 peticiones a la vez, realizando un POST al proyecto deployado en Render.



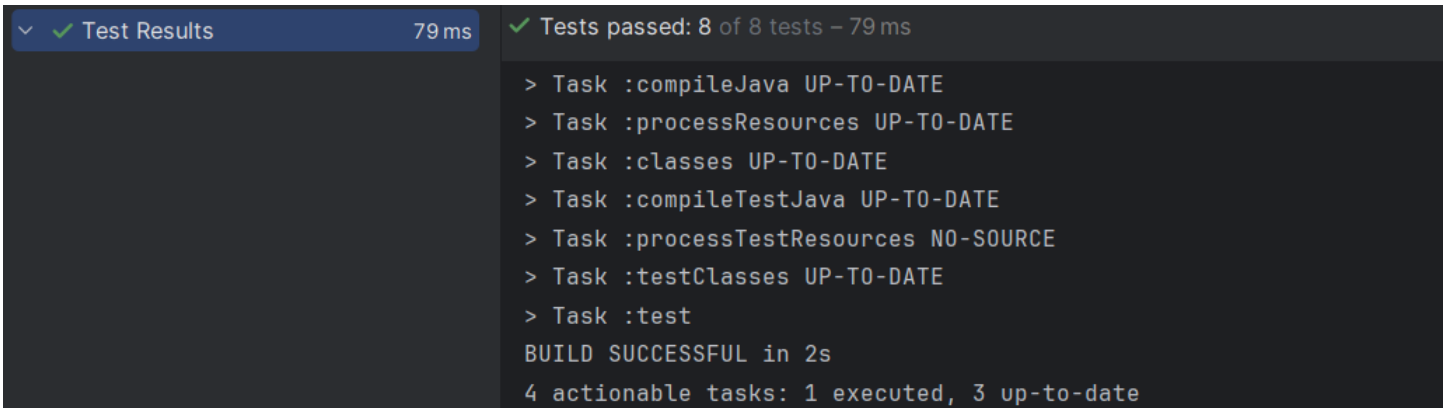


JUNIT – PRUEBAS UNITARIAS

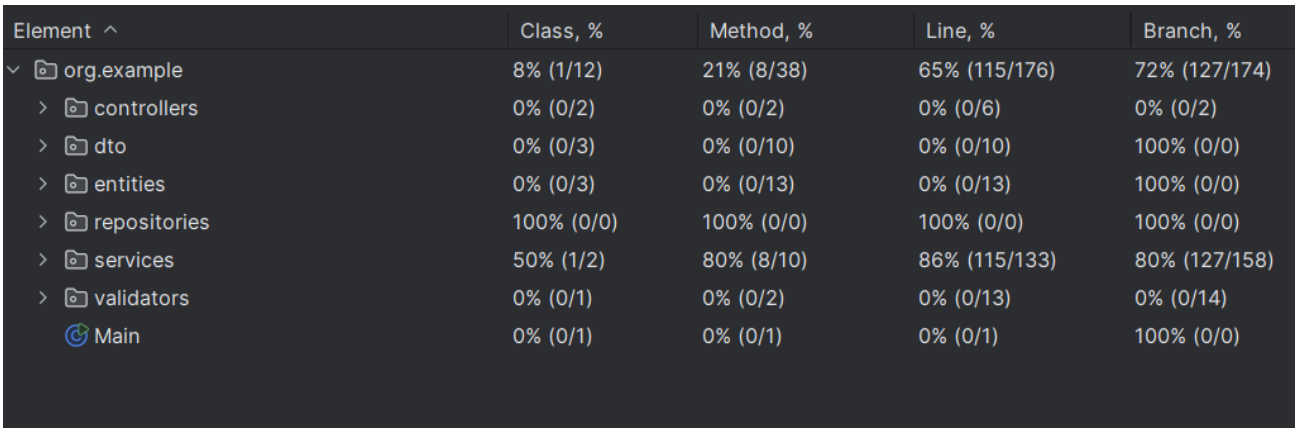
El programa cuenta con pruebas unitarias realizadas con Junit. Se realizaron 8 pruebas unitarias, las cuales se encuentran en src/test/java/DnaServiceTest.

Luego de ejecutar las pruebas, estos fueron los resultados de los test y del Code Coverage.

RESULTADO DE LOS TEST



CODE COVERAGE



Element	Class, %	Method, %	Line, %	Branch, %
org.example	8% (1/12)	21% (8/38)	65% (115/176)	72% (127/174)
controllers	0% (0/2)	0% (0/2)	0% (0/6)	0% (0/2)
dto	0% (0/3)	0% (0/10)	0% (0/10)	100% (0/0)
entities	0% (0/3)	0% (0/13)	0% (0/13)	100% (0/0)
repositories	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
services	50% (1/2)	80% (8/10)	86% (115/133)	80% (127/158)
validators	0% (0/1)	0% (0/2)	0% (0/13)	0% (0/14)
Main	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)