

NIVEL 3:

H2:

La base de datos elegida es H2, lo que buscamos es que cada secuencia que se le ingrese, el sistema verifique si es mutante o no, y que esa secuencia se guarde en H2.

jdbc:h2:mem:inicial1

DNA

INFORMATION_SCHEMA

Users

H2 2.2.224 (2023-09-17)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM DNA

SELECT * FROM DNA;

ID	DNA_SEQUENCE	IS_MUTANT
1	ATGCGAAAAAGCTTGTGTAGTTTGCTCCGATCACTG	FALSE
2	ATGCGAAATAGCTTGTGTAGTTGGCTCCGATCACTG	TRUE
3	ATGCGAAATAGCTTGTGTAGTATGCTCCGATCACTG	TRUE
4	ATGCGAAATAGCTTGTGTAGTGTGCTCCGATCACTG	TRUE
5	TTTTGAAATAGCTTGAGTAGTGTGCTCCGATCACTG	TRUE
6	TATTGAAATAGCTTGAGTAGTGTGCTCCGATCACTG	FALSE
7	TATTGAAATAGCTTGAGTATTGTGCTCCGATCACTG	FALSE
8	TATTGAAGTAGCTTGAGTATTGTGCTCCGATCACTA	FALSE
9	TATTGAAGTAGCTTTTGTATTGTGCTCCGATCACTA	TRUE
10	ATGCGAAGAAGCTTGTGTAGTGTGCTCCGATCACTG	TRUE
11	ATGCGAAGAAGCTTTTGTAGTGTGCTCCGATCACTG	FALSE
12	AAAAGAAGAAGCTTTTGTAGTGTGCTCCGATCACTG	TRUE
13	AAAAGAAGAAGCTTATGTAGTGTGCTCCGATCACTG	FALSE
14	ATAAGAATAAGCTTATGTATTGTGCTCCGATCACTG	TRUE
15	ATAAGAATAAGCTTATGTATTGTGCACCGATCACTG	FALSE
16	ATAAGAACAAGCTTATGTATTGTGCACCGATCACTG	FALSE
17	ATAAGAACAAGCATATGTATTGTGCACCGATCACTG	FALSE
18	ATAAGAACAAGCATATGTATTGTGAACCGATCACTG	TRUE

(18 rows, 0 ms)

Edit

CODE COVERAGE

Overall Coverage Summary				
Package	Class, %	Method, %	Branch, %	Line, %
all classes	87,5% (7/8)	85,7% (18/21)	92,9% (65/70)	93,8% (75/80)
Coverage Breakdown				
Package	Class, %	Method, %	Branch, %	Line, %
com.example.inicial1	100% (1/1)	50% (1/2)		33,3% (1/3)
com.example.inicial1.controllers	100% (2/2)	100% (6/6)	92,9% (26/28)	97% (32/33)
com.example.inicial1.dto	100% (1/1)	66,7% (2/3)		75% (3/4)
com.example.inicial1.entities	50% (1/2)	66,7% (2/3)		66,7% (2/3)
com.example.inicial1.exception	100% (1/1)	100% (1/1)		100% (1/1)
com.example.inicial1.services	100% (1/1)	100% (6/6)	92,9% (39/42)	100% (36/36)

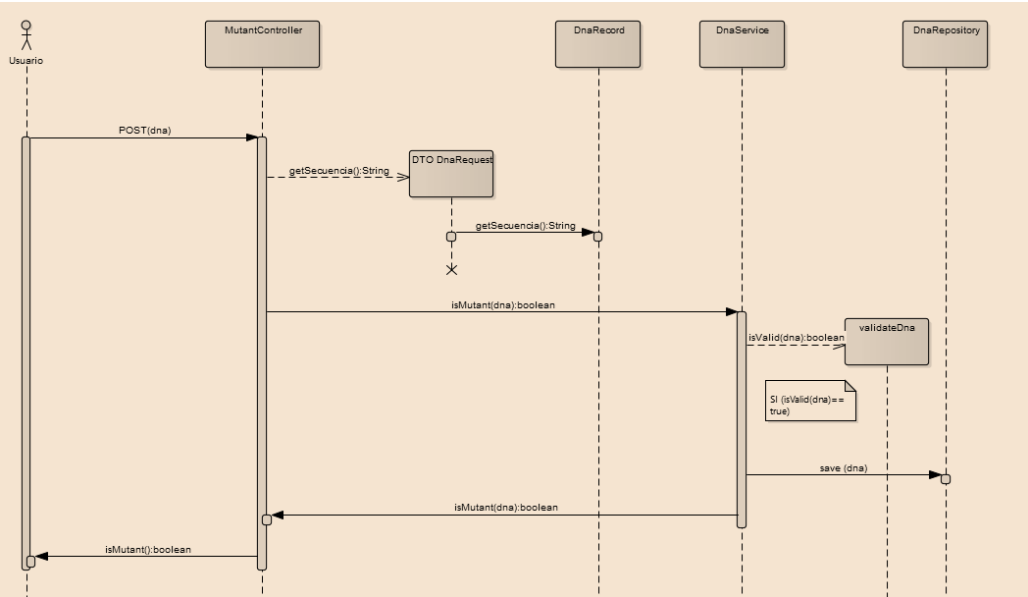
Cobertura General

- Cobertura de línea: El 93.8% de las líneas de código están cubiertas por pruebas. Este es un porcentaje bastante alto y sugiere una buena cobertura.
- Cobertura de método: El 85.7% de los métodos están cubiertos por pruebas. Si bien es un buen porcentaje, podría mejorarse en algunos paquetes.
- Cobertura de rama: El 92.9% de las ramas están cubiertas por pruebas. Esto indica que las diferentes rutas de ejecución en el código están siendo probadas de manera exhaustiva.
- Cobertura de clase: El 87.5% de las clases están cubiertas por pruebas. Este porcentaje también es bastante bueno.

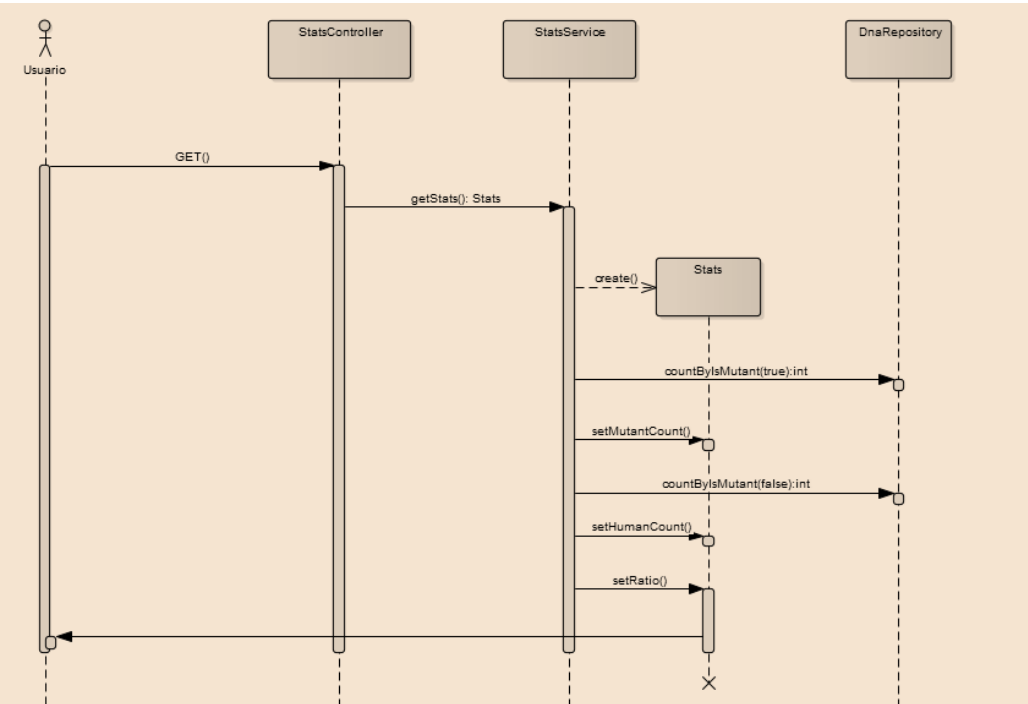
En general, se cumple con el objetivo de una cobertura de código superior al 80%. La cobertura de línea, rama y clase es bastante alta, lo que indica que mi código está bien probado. Sin embargo, hay áreas específicas (como algunos métodos en ciertos paquetes) donde se podría mejorar la cobertura.

DIAGRAMAS DE SECUENCIAS

POST→Mutant

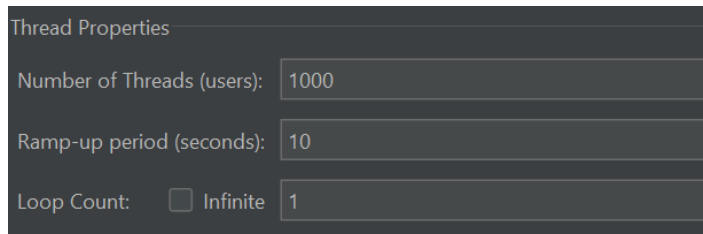


GET → STATS



Pruebas de Stress(Jmeter)

Las pruebas de stress permiten verificar cómo se comporta la aplicación cuando un número elevado de usuarios accede simultáneamente a los servicios. Esto ayuda a identificar el límite de usuarios que el sistema puede manejar sin degradar su rendimiento.



Thread Properties

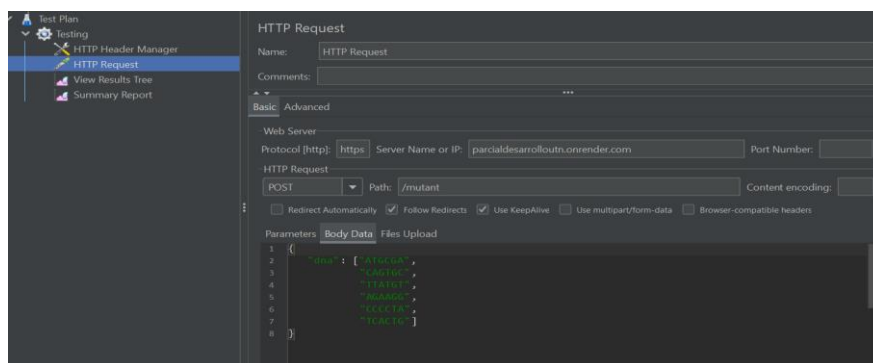
Number of Threads (users): 1000

Ramp-up period (seconds): 10

Loop Count: ☐ Infinite 1

Se configuro el para 1000 peticiones en un periodo de 10 seg.

Para **POST** → mutant, utilizando lo generado con render:
“https://parcialdesarrolloutn.onrender.com/mutant”



HTTP Request

Name: HTTP Request

Comments:

Basic Advanced

Web Server

Protocol (http): https Server Name or IP: parcialdesarrolloutn.onrender.com Port Number:

HTTP Request

POST Path: /mutant Content encoding:

☐ Redirect Automatically ☒ Follow Redirects ☒ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Parameters Body Data Files Upload

1 {

2 "area": [

3 "1A50-8A",

4 "1A5100",

5 "1A5101",

6 "A0A000",

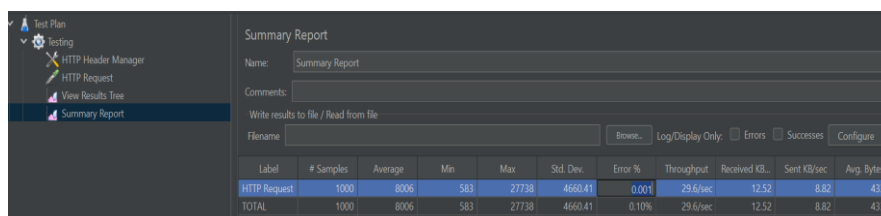
7 "1A50-10",

8 "1A50-10"

9]

10 }

Configuramos nuestro **HTTP Request**, como se ve en la imagen.



Summary Report

Name: Summary Report

Comments:

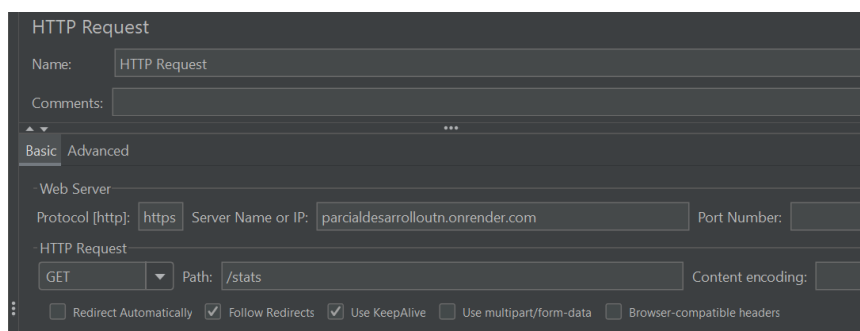
Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB...	Sent KB/sec	Avg. Bytes
HTTP Request	1000	8006	583	27738	4660.41	0.00%	29.6/sec	12.52	8.82	433.7
TOTAL	1000	8006	583	27738	4660.41	0.10%	29.6/sec	12.52	8.82	433.7

Notamos que al recibir 1000 peticiones tenemos un error de 0,001

Para **GET** → stats, utilizando lo generado con render:
“https://parcialdesarrolloutn.onrender.com/stats”



HTTP Request

Name: HTTP Request

Comments:

Basic Advanced

Web Server

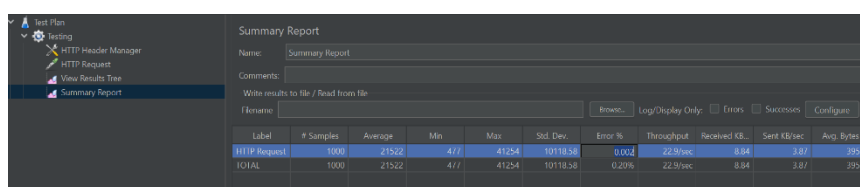
Protocol (http): https Server Name or IP: parcialdesarrolloutn.onrender.com Port Number:

HTTP Request

GET Path: /stats Content encoding:

☐ Redirect Automatically ☒ Follow Redirects ☒ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Configuramos nuestro **HTTP Request**, como se ve en la imagen.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB...	Sent KB/sec	Avg. Bytes
HTTP Request	1000	21522	417	41454	10118.38	0.00%	22.9/sec	8.94	3.87	293.1
TOTAL	1000	21522	417	41454	10118.38	0.29%	22.9/sec	8.94	3.87	293.1

Notamos que al recibir 1000 peticiones tenemos un error de 0,002

