Documentación
Test Práctico - Mercado Libre

Autor: Lucas Miguel Avila

URL API : https://challenge-mercadolibre.herokuapp.com/
GITHUB: https://github.com/lucasavila/challenge-mercado-libre

Índice:

Información de la API REST

La API REST se encuentra implementada en el host: https://challenge-mercadolibre.herokuapp.com/ de Cloud Application Platform | Heroku

El código fuente se encuentra en github: https://github.com/lucasavila/challenge-mercado-libre

La misma utiliza una Base de Datos Postgres también alojada en Cloud Application Platform | Heroku.

Se publicaron las 2 URLS comprendidas en el Test:

- Para ejecutar el cálculo de si es el ADN mutante.

  POST
  https://challenge-mercadolibre.herokuapp.com/mutant/

  El cual espera un body/json con el siguiente formato:

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

- Para devolver las estadísticas de la ejecución

  GET
  https://challenge-mercadolibre.herokuapp.com/stats/

Descripción del Algoritmo Utilizado

Evalúa en el array de Strings si alguno de los valores no corresponde a una secuencia de ADN válidos. Esto se valida con una expresión regular.

Se utilizó un algoritmo programático el cual recorre de a uno todas opciones de ser posible pero al encontrar 2 combinaciones (cantidad de combinaciones necesarias para ser mutante) el mismo ejecuta un corte de control evitando realizar más cálculos innecesarios.

La matriz es recorrida y calcula en caso de ser posible las combinaciones en columna, fila, diagonal descendente y diagonal ascendente.

Se guardan y se obtienen los registros de la base de datos con JPA con un repository service.

(Si se quiere el guardado inmediato en la base de datos se encuentra el desarrollo en el **branch release)**

Mejoras realizadas: Ya que no se indica que el impacto en la base de datos debe ser inmediato, se agregó un framework de actores GPars de Groovy para poder realizar la insercion de la base de datos de manera descentralizada, focalizándose en el cálculo de mutantes para su pronta respuesta.

Instrucciones de Ejecución

Previo a la ejecución, se debe tener instalado Docker (preferentemente versiones recientes como la 18.02) y java 8.

Para la ejecución localmente se generó un script el cual se ejecuta de la siguiente manera:

bash runApp.sh

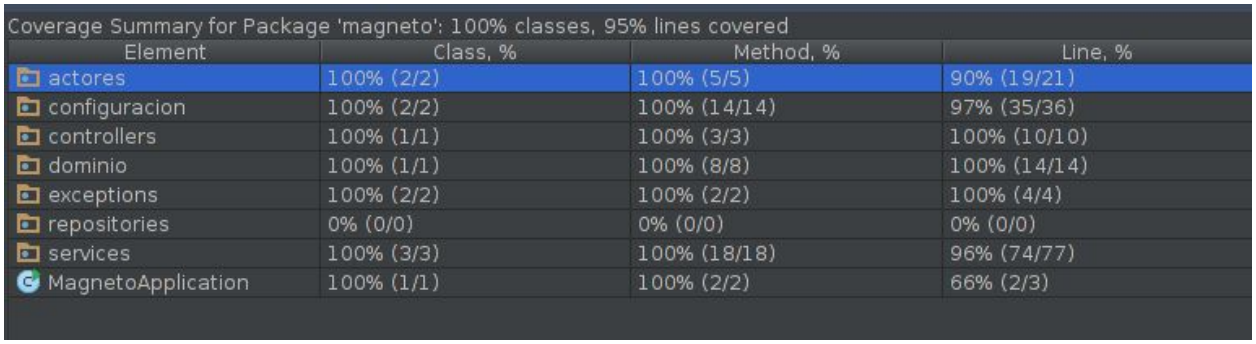
El mismo ejecuta un script llamado db.sh que descarga una imagen postgres 9.4 de Docker y levanta el container con la misma.

Luego, se ejecuta el bootRun de Gradle (se agrego un gradle wrapper para que pueda desplegar automáticamente) el cual tiene como parámetro “create” en el schema de hibernate para poder agregar la tabla necesaria para la ejecución del algoritmo.

Pruebas Realizadas

Test Automatizados

Se realizaron Test Unitarios con **Spockframework** y de Integración con Junit con un 95% de líneas cubiertas.
(También se genero un script llamado **runTests.sh** el cual ejecuta los Tests desde consola.)



Coverage Summary for Package 'magneto': 100% classes, 95% lines covered

Element	Class, %	Method, %	Line, %
actores	100% (2/2)	100% (5/5)	90% (19/21)
configuracion	100% (2/2)	100% (14/14)	97% (35/36)
controllers	100% (1/1)	100% (3/3)	100% (10/10)
dominio	100% (1/1)	100% (8/8)	100% (14/14)
exceptions	100% (2/2)	100% (2/2)	100% (4/4)
repositories	0% (0/0)	0% (0/0)	0% (0/0)
services	100% (3/3)	100% (18/18)	96% (74/77)
MagnetoApplication	100% (1/1)	100% (2/2)	66% (2/3)

Pruebas de Estrés (JMETER)

Las siguientes pruebas de estrés fueron realizadas con la herramienta JMETER con un **Ramp-up Period** de 1 segundo y **Number of threads** 10 se ejecutaron 10000 las cuales arrojaron un resultado de 100000 Samples

Se utilizó el siguiente JSON de 10x10 que se verifica que es mutante

```
{
"dna":[
    "ATGCGATAGC",
    "CAGTGCATGC",
    "TTATGTGCTA",
    "AGAAGGTATA",
    "CCCCTAGAAA",
    "TAGGATAGAG",
    "TCACTGTTGG",
    "ATAATGTATA",
    "TTGGGATAGC",
    "TGAGATGCTA"
    ]
}
```

Originalmente se ejecutaron en **29** segundos con un throughput de **3341**/seg.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received ...	Sent KB/sec
HTTP Req...	100000	2	2	5	7	15	1	204	0.00%	3341.1/sec	450.89	1318.18
TOTAL	100000	2	2	5	7	15	1	204	0.00%	3341.1/sec	450.89	1318.18

Luego de realizar la mejora implementando el framework de actores de groovy:

Se ejecutaron en **13** segundos y se incrementó el throughput de **3341**/seg a **9161**/seg

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received ...	Sent KB/sec
HTTP Req...	100000	0	1	2	3	9	0	201	0.00%	9161.7/sec	1236.38	3668.26
TOTAL	100000	0	1	2	3	9	0	201	0.00%	9161.7/sec	1236.38	3668.26

Se utilizó el siguiente JSON de 10x10 que se verifica que **NO** es mutante

```
{
"dna":[
    "ACACACACAC",
    "CACACACACA",
    "TGTGTGTGTG",
    "GTGTGTGTGT",
    "ACACACACAC",
    "GTGTGTGGTG",
    "TGTGTGTGTG",
    "ACACACACAC",
    "ACACACACAC",
    "TGTGTGTGTG"
    ]
}
```

Originalmente se ejecutaron en **31** segundos con un throughput de **3171**/seg.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received K...	Sent KB/sec
HTTP Request	100000	3	2	5	7	14	1	192	100.00%	3171.5/sec	966.90	1251.25
TOTAL	100000	3	2	5	7	14	1	192	100.00%	3171.5/sec	966.90	1251.25

Luego de realizar la mejora implementando el framework de actores de groovy:

Se ejecutaron en **17** segundos y se incrementó el throughput de **3171**/seg a **6757**/seg

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received ...	Sent KB/sec
HTTP Req...	100000	1	1	2	4	10	0	236	100.00%	6757.7/sec	2060.23	2666.11
TOTAL	100000	1	1	2	4	10	0	236	100.00%	6757.7/sec	2060.23	2666.11

Se generó el archivo **mercado-libre.jmx** en el cual se guarda la información para la corrida, el mismo se encuentra en el repositorio de github.

Referencias

Heroku: www.heroku.com
Github: https://github.com/
Docker: https://www.docker.com/
Postgres: https://www.postgresql.org/
Gradle: https://gradle.org/
Java: https://www.java.com/
Groovy: http://groovy-lang.org/
GPars: http://gpars.org/
Spock: http://spockframework.org/
JMeter: https://jmeter.apache.org/