Examen Mercado Libre

Autor: Matias Garcia Marset
Fecha: 14/02/2018

Asunciones:

- El proyecto va a escalar, la cantidad de tipos de ADNs puede aumentar.
- En el nivel 3 habla de que la API puede recibir fluctuaciones. Entendí que las mismas son sobre las estadísticas. Esto es importante porque la decisión de usar una cache aca fue en base a que no se modifica tanto la base de datos (usando /mutant). Esto lo explico en Nivel 2/3
- En el nivel 2 solo se pueden devolver 200 (si es mutante) o 403 en cualquier otro caso.
- El ratio es la división entre cantidad de humanos y mutantes (el que tenga menor cantidad sobre el que tiene mayor, para que sea numero entre 0 y 1).

Instrucciones para levantar el proyecto localmente

- Bajar repositorio git: https://github.com/matiasgarciamarset/ML-Request
- Tener eclipse Oxygen 2 con java 1.8 con el plugin TestNG y Google Engine (opcional).
- Importar proyecto
- Para correrlo localmente con tomcat ejecutar: mvn tomcat7:run
- Para correr los test: Click derecho sobre la clase con los tests y poner Run with TestNG.

APIs

Nivel 2 (Post): <URL>/api/analyze/mutant

Ejemplo de body: {"dna":["ATGCGA","CAGTGC","TTATGT","AGAAGG","CACCTA","TCACTG"]}

Nivel 3 (Get): <URL>/api/analyze/stats

Donde <URL> = http://35.198.58.36/ml%2Drequest *ó* https://ml-request.appspot.com

---

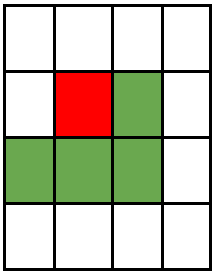
Calidad

- En el proyecto se adjuntan tests unitarios (AnalizadorServiceImplTest.java). Para correr los mismos en eclipse, se requiere el plugin de testNG.
- Idealmente se deberían agregar tests de integración que testeen la API y los DAOs.
- Adjunto regresiones al final del documento.

Nivel 1

Explicación solución:

La idea es ir recorriendo de izquierda a derecha y de arriba hacia abajo la matriz.
En cada paso lo que se hace es “decirle” al vecino que tiene la misma letra, que hay un camino que llega hasta él con un determinado largo. Es decir, si la posición actual es el cuadrado rojo, entonces los vecinos son los verdes:

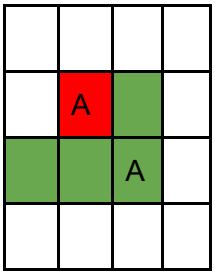


Osea, el que esta:

- A la derecha
- Abajo a la izquierda
- Abajo
- ó Abajo a la derecha

es un vecino.

De esta manera, supongamos que el de abajo a la derecha tiene la misma letra que el actual



Entonces al de abajo a la derecha le damos nuestra cantidad acumulada (de posiciones que encontramos “abajo a la derecha”) y le sumamos uno.

De esta manera, cuando una de las posiciones llega a 4, encontramos una secuencia horizontal, vertical ó diagonal que contiene cuatro letras iguales consecutivas.

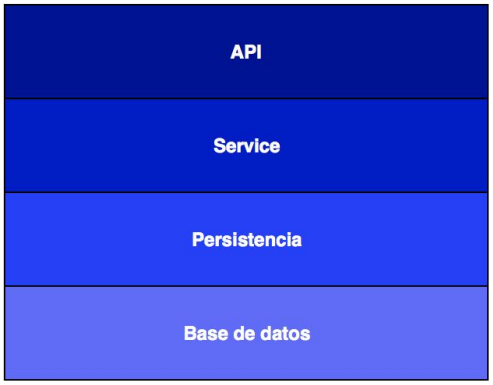
Complejidad

Dado que solo recorremos la matriz una vez (NxN) y en cada iteraciones asignamos valores a los vecinos (costo constante); El costo total de la búsqueda es O(NxN), es decir, lineal.

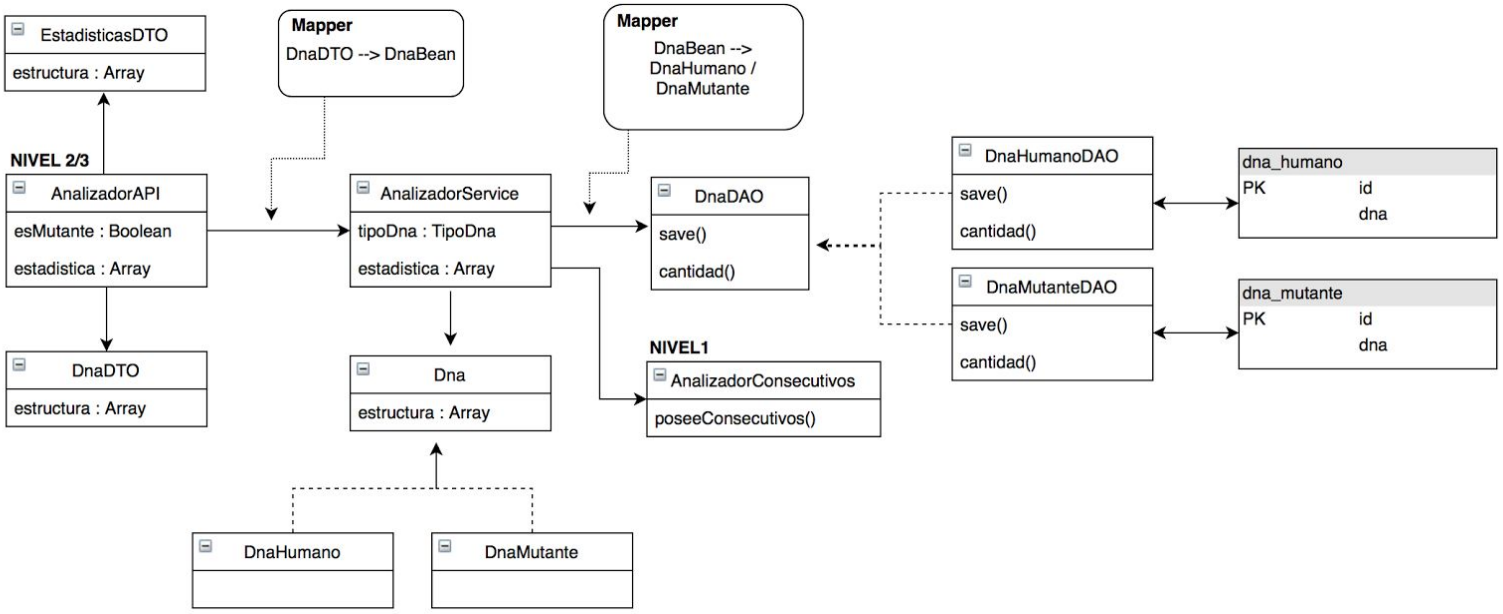
Y en menor complejidad no se puede realizar ya que si o si hay que recorrer todos los valores (esto no quita que se puedan hacer mejoras de performance, solo hago alusión a la complejidad asintótica).

Nivel 2/3

Para realizar estos niveles decidí usar Java 8 junto con Spring y Hibernate. La arquitectura del proyecto quedó por capas:



de esta manera:

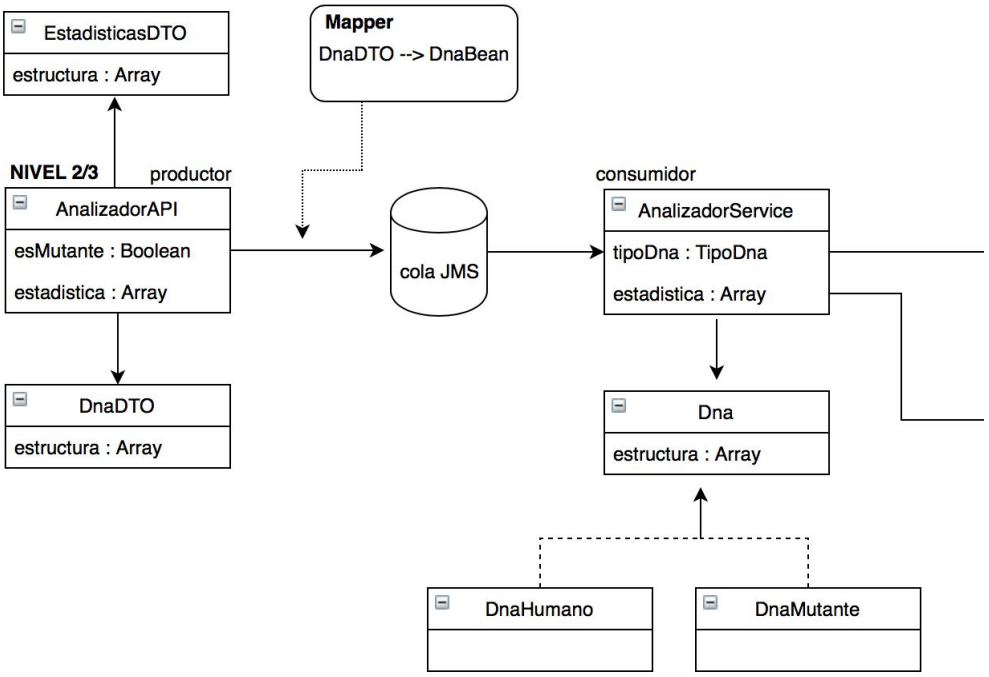


Aquí una decisión importante fue la de dividir Dna en Humano y Mutante. La misma fue tomada pensando en optimizar la consulta de estadísticas, ya que la misma requiere la cantidad total de registros de ambos, y tener divididos los adn en dos tablas implica que la BD puede mantener registro de la cantidad de entradas y retornarlas mucho más rápido que si tiene que hacer un count sobre una query.

A su vez, para hacer más eficiente el pedido de estadísticas agregue una cache sobre la query que cuenta la cantidad de entradas de las tablas (ya que la consulta a la base de datos suele ser algo costoso).

Una experimentación que quedó pendiente es agregar una cola JMS luego del endpoint para poder asegurar mejor disponibilidad. Digo experimentación porque dado que no hay

mucha complejidad en la llamada a /stats puede resultar que el hecho de poner el mensaje en una cola baje la performance.



Para esto se podría utilizar Camel como framework.

Integración con Google engine

Para esto existen varias posibilidades, entre otras:

1) Crear un proyecto maven de cero usando el architype de google. (https://cloud.google.com/appengine/docs/standard/java/tools/maven)
2) Migrar un proyecto maven existente para usar Google engine. (https://cloud.google.com/appengine/docs/standard/java/tools/maven)
3) Crear un servidor tomcat en una VM (https://console.cloud.google.com/launcher/details/click-to-deploy-images/tomcat)

Migrar un proyecto maven existente para usar Google engine

1) Agregar el plugin de Google Engine a eclipse.
2) instalar sdk: https://cloud.google.com/sdk/docs/
3) Correr ./google-cloud-sdk/bin/gcloud components install app-engine-java
4) Migrar proyecto a google engine app (https://cloud.google.com/eclipse/docs/migrating-gpe)

5) En eclipse ir a preferencias -> Google Cloud Tools y seleccionar la carpeta del sdk
6) Crear instancia de MySql en google: https://cloud.google.com/sql/docs/mysql/quickstart
7) Tambien hay que habilitar las APIs creadas: https://cloud.google.com/endpoints/docs/openapi/deploy-api-backend

Lo bueno de la primera y segunda opciones es que al usar el sdk de google permite manejar muchísimas cosas de plataform (como el balanceo de carga, crear instancias dinámicamente, manejar estadísticas de apis, etc). Por ejemplo:

- Configurar un ambiente escalable: https://cloud.google.com/appengine/docs/flexible/

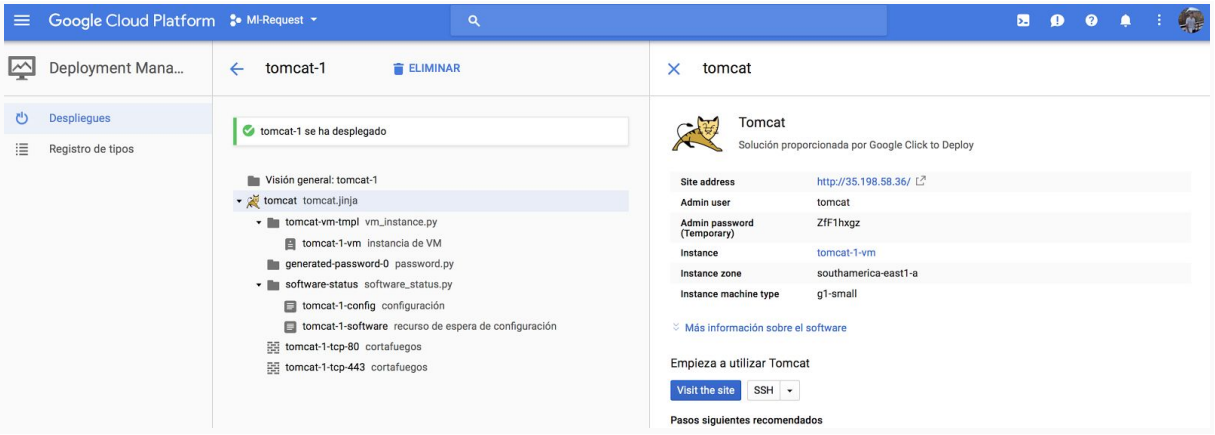
Tener en cuenta que estas soluciones suelen ser bastante costosas (económicamente hablando).

Crear un servidor tomcat en una VM

(Esta opción es la que termine utilizando)

1) Crear un war del proyecto (En eclipse click derecho sobre el proyecto y “exportar”)
2) Crear una instancia de tomcat https://cloud.google.com/appengine/docs/standard/java/tools/maven

Aquí se pueden elegir muchas opciones de VM (yo elegí la mas barata y con menos recursos).



consola administrativa: http://35.198.58.36/manager/html

2) Autorizar la ip 35.198.58.36 para que pueda acceder a la instancia de MySql
3) Abrir la consola administrativa de tomcat, deployar el war y poner “start”.

Regresiones

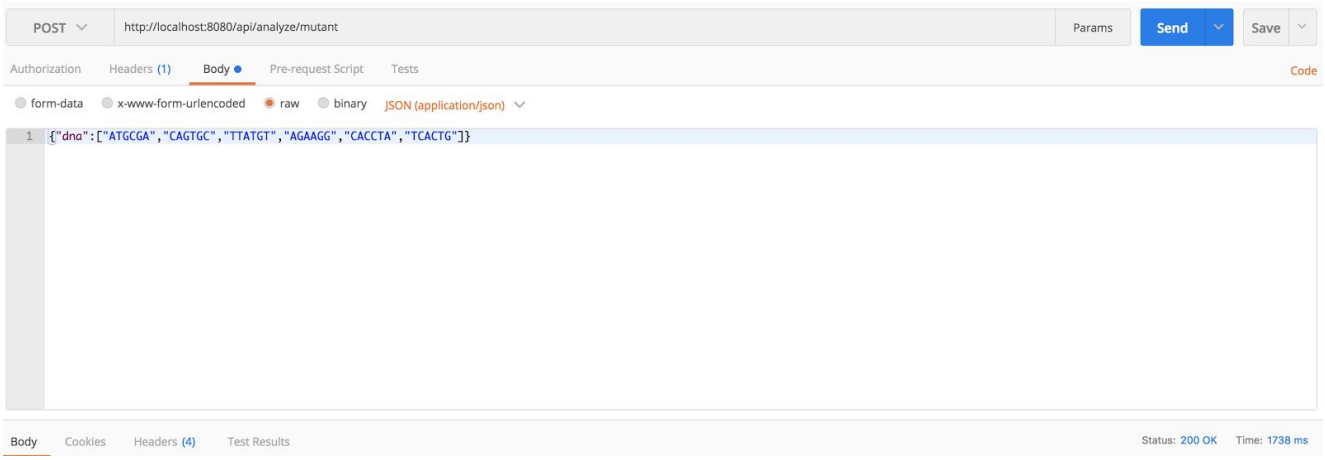
- *Como usuario puedo detectar si un adn mutante es efectivamente mutante*

Requiere: -
Pasos:

1) Entrar en postman

2) Seleccionar POST, URL=http://localhost:8080/api/analyze/mutant y BODY= {"dna":["ATGCGA","CAGTGC","TTATGT","AGAAGG","CACCTA","TCACTG"]}
3) Click en “Send”

Comportamiento esperado: Status 200 OK



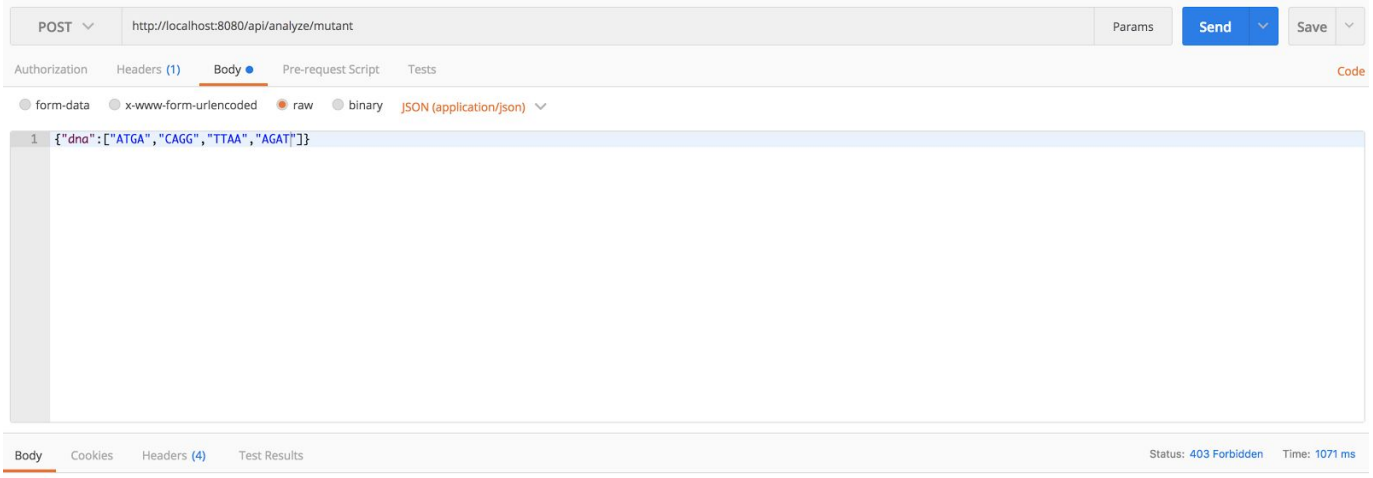
- *Como usuario puedo detectar si un adn humano es efectivamente humano*

Requiere: -

Pasos:

1) Entrar en postman
2) Seleccionar POST, URL=http://localhost:8080/api/analyze/mutant y BODY= {"dna":["ATGA","CAGG","TTAA","AGAT"]}
3) Click en “Send”

Comportamiento esperado: Status 403 forbidden



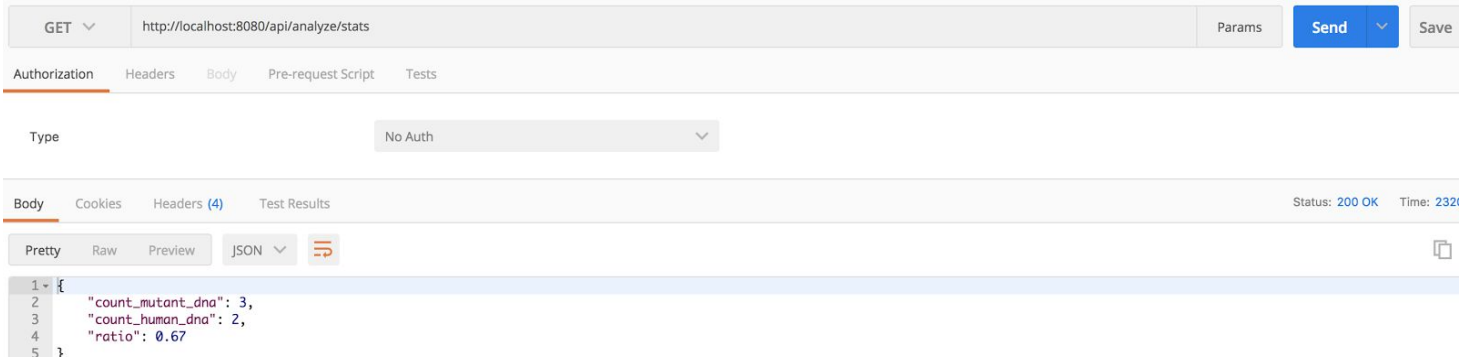
- *Como usuario puedo acceder a las estadísticas de adn*

Requiere: -

Pasos:

1) Entrar en postman
2) Seleccionar GET, URL=http://localhost:8080/api/analyze/stats
3) Click en “Send”

Comportamiento esperado: Se deben visualizar los resultados correspondientes a la cantidad de registros de las tablas dna_humano y dna_mutante.


GET
http://localhost:8080/api/analyze/stats
Params
Send
Save
Authorization
Headers
Body
Pre-request Script
Tests
Type
No Auth
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 232
Pretty
Raw
Preview
JSON
{
"count_mutant_dna": 3,
"count_human_dna": 2,
"ratio": 0.67
}