Análisis Challenge Mercado Libre Detección Mutantes	
Autor:	JORGE DAVID QUIROGA ROJAS
Fecha:	23/03/2021

La solicitud de solución para mercado libre requería dos APIs para su publicación final.

Api **/mutants/{dna}**

Nivel 1:

El api mutants requiere la validación de una cadena de caracteres para evaluar si la repetición de un carácter en 4 o más posiciones consecutivas clasifica al ADN como mutante. Se recibe un arreglo de strings con los datos a evaluar.

Lo primero que hago es transformar ese arreglo en un objeto más volátil para realizar la evaluación:

```
/**
 *
 * @author usuario
 */
@ApiModel
@Data
@ToString
@NoArgsConstructor
@AllArgsConstructor
public class PosicionMatrizVO {

    private int fila;
    private int columna;
    private String valor;

}
```

Este objeto permite identificar un valor en una posición especifica de la matriz. En representación con el ejemplo indicado por el challenge, se guarda una colección de este tipo de objeto con la siguiente estructura:

String[] dna = {"ATGCGA","CAGTGC","TTATGT","AGAAGG","CCCCTA","TCACTG"};

Esto transformado a la matriz se representaría de la siguiente manera:

	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

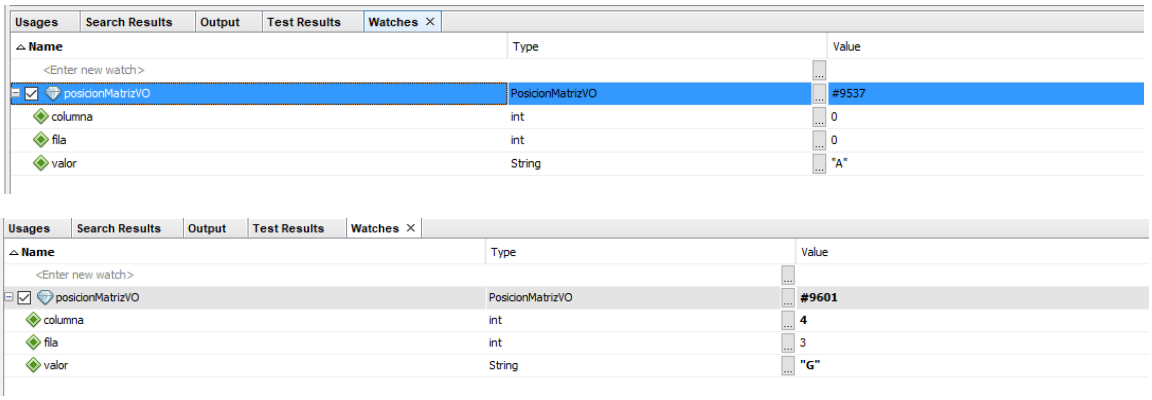
Donde las filas están representadas en las posiciones de izquierda a derecha. Las columnas están representadas de arriba hacia abajo. Para este caso, cada carácter va a estar asociado en el objeto tiene una coordenada y un valor representado en la letra.

Por ejemplo:

	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Para el carácter A resaltado en naranja, su coordenada en la matriz se encuentra en [fila=0] y [columna=0], mientras que el carácter G resaltado en verde, su coordenada en la matriz se encuentra [fila=3] y [columna=4].

La representación del objeto en el programa es la siguiente:



Antes de comenzar la evaluación, es requerido identificar unas restricciones implícitas y explicitas en la solicitud de la solución:

- Que la matriz sea cuadrada (Tipo [NxN])

  Esta validación es primordial, debido que es requerido que cumpla esta estructura para poder efectuar cualquier validación. Esto se realiza verificando la cantidad de elementos del arreglo vs la cantidad de caracteres que contiene cada elemento de ese arreglo.

  Por ejemplo, para el caso indicado en el challenge:

  String[] dna = {"ATGCGA","CAGTGC","TTATGT","AGAAGG","CCCCTA","TCACTG"};

  Encontramos que el arreglo tiene 6 elementos:

  { "ATGCGA" , "CAGTGC" , "TTATGT" , "AGAAGG" , "CCCCTA" , "TCACTG" }
  1 2 3 4 5 6

  Y que cada elemento contiene 6 caracteres.

  A T G C G A
  1 2 3 4 5 6

Esta evaluación se debe realizar por cada elemento. Para este caso, todos los elementos tienen 6 caracteres por lo tanto se cumple la regla de matriz cuadrada, esta específicamente es de [6x6]. En caso que no sea de este tipo, el sistema indicará una excepción por estructura invalida.

Esta validación se ejecuta en la siguiente parte del código:

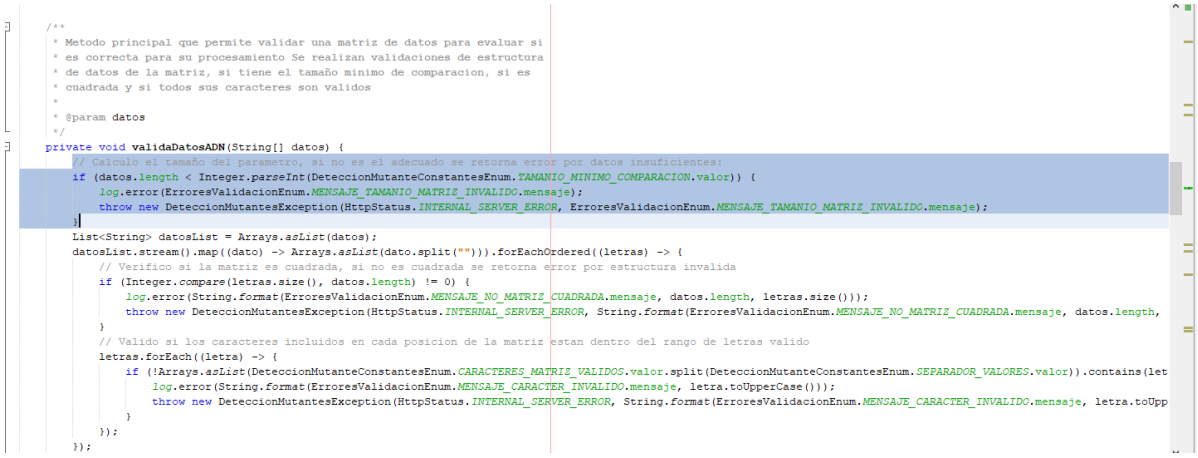
```
/**
 * Metodo principal que permite validar una matriz de datos para evaluar si
 * es correcta para su procesamiento Se realizan validaciones de estructura
 * de datos de la matriz, si tiene el tamaño minimo de comparacion, si es
 * cuadrada y si todos sus caracteres son validos
 *
 * @param datos
 */
private void validaDatosADN(String[] datos) {
    // Calculo el tamaño del parametro, si no es el adecuado se retorna error por datos insuficientes:
    if (datos.length < Integer.parseInt(DeteccionMutanteConstantesEnum.TAMANIO_MINIMO_COMPARACION.valor)) {
        log.error(ErroresValidacionEnum.MENSAJE_TAMANIO_MATRIZ_INVALIDO.mensaje);
        throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, ErroresValidacionEnum.MENSAJE_TAMANIO_MATRIZ_INVALIDO.mensaje);
    }
    List<String> datosList = Arrays.asList(datos);
    datosList.stream().map((dato) -> Arrays.asList(dato.split(""))).forEachOrdered((letras) -> {
        // Verifico si la matriz es cuadrada, si no es cuadrada se retorna error por estructura invalida
        if (Integer.compare(letras.size(), datos.length) != 0) {
            log.error(String.format(ErroresValidacionEnum.MENSAJE_NO_MATRIZ_CUADRADA.mensaje, datos.length, letras.size()));
            throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, String.format(ErroresValidacionEnum.MENSAJE_NO_MATRIZ_CUADRADA.mensaje, datos.length,
        }
        // Valido si los caracteres incluidos en cada posicion de la matriz estan dentro del rango de letras valido
        letras.forEach((letra) -> {
            if (!Arrays.asList(DeteccionMutanteConstantesEnum.CARACTERES_MATRIZ_VALIDOS.valor.split(DeteccionMutanteConstantesEnum.SEPARADOR_VALORES.valor)).contains(let
                log.error(String.format(ErroresValidacionEnum.MENSAJE_CARACTER_INVALIDO.mensaje, letra.toUpperCase()));
                throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, String.format(ErroresValidacionEnum.MENSAJE_CARACTER_INVALIDO.mensaje, letra.toUpp
            }
        });
    });
```

Donde letras.size() corresponde a la cantidad de caracteres del elemento y datos.length corresponde a la cantidad de atributos en el arreglo.

- Que la matriz tenga el tamaño mínimo de validación.

Uno de los requerimientos base de la solicitud es que, para que sea un mutante, debe repetirse un carácter al menos cuatro (4) veces en horizontal, vertical u oblicuo. Por lo tanto, para que la matriz sea candidata a validación, debe tener un tamaño minimo de 4x4. De lo contrario, se genera una excepción por datos insuficientes. Tambien por entendimiento podría caracterizarse que el ADN es de un humano, pero para este caso de implementación preferí generar la excepción indicada.

Esta validación se ejecuta en la siguiente parte del código:



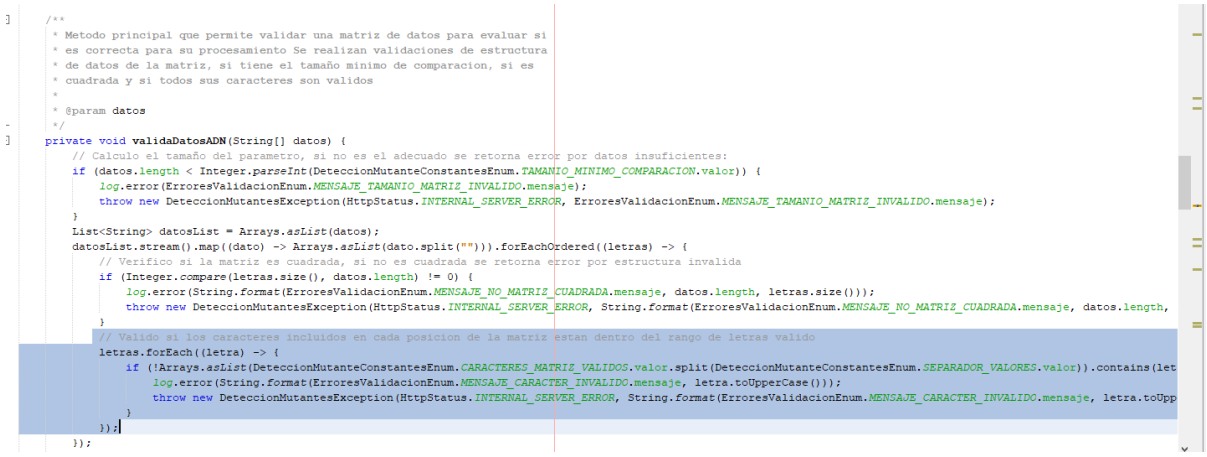
```
/**
 * Metodo principal que permite validar una matriz de datos para evaluar si
 * es correcta para su procesamiento Se realizan validaciones de estructura
 * de datos de la matriz, si tiene el tamaño minimo de comparacion, si es
 * cuadrada y si todos sus caracteres son validos
 *
 * @param datos
 */
private void validaDatosADN(String[] datos) {
    // Calculo el tamaño del parametro, si no es el adecuado se retorna error por datos insuficientes:
    if (datos.length < Integer.parseInt(DeteccionMutanteConstantesEnum.TAMANIO_MINIMO_COMPARACION.valor)) {
        log.error(ErroresValidacionEnum.MENSAJE_TAMANIO_MATRIZ_INVALIDO.mensaje);
        throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, ErroresValidacionEnum.MENSAJE_TAMANIO_MATRIZ_INVALIDO.mensaje);
    }
    List<String> datosList = Arrays.asList(datos);
    datosList.stream().map((dato) -> Arrays.asList(dato.split(""))).forEachOrdered((letras) -> {
        // Verifico si la matriz es cuadrada, si no es cuadrada se retorna error por estructura invalida
        if (Integer.compare(letras.size(), datos.length) != 0) {
            log.error(String.format(ErroresValidacionEnum.MENSAJE_NO_MATRIZ_CUADRADA.mensaje, datos.length, letras.size()));
            throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, String.format(ErroresValidacionEnum.MENSAJE_NO_MATRIZ_CUADRADA.mensaje, datos.length,
        }
        // Valido si los caracteres incluidos en cada posicion de la matriz estan dentro del rango de letras valido
        letras.forEach((letra) -> {
            if (!Arrays.asList(DeteccionMutanteConstantesEnum.CARACTERES_MATRIZ_VALIDOS.valor.split(DeteccionMutanteConstantesEnum.SEPARADOR_VALORES.valor)).contains(let
                log.error(String.format(ErroresValidacionEnum.MENSAJE_CARACTER_INVALIDO.mensaje, letra.toUpperCase()));
                throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, String.format(ErroresValidacionEnum.MENSAJE_CARACTER_INVALIDO.mensaje, letra.toUpp
            }
        });
    });
```

Donde datos.length corresponde a la cantidad de elementos del arreglo, en caso que sea menor a cuatro (4), se dispara la excepción y finaliza la rutina.

- Que los caracteres que se ingresan se encuentren en el rango valido solicitado

Al ser un parámetro alfanumérico, el programa recibe cualquier carácter en su invocación. La solicitud explicita indica que los caracteres correspondientes a una cadena de ADN solo pueden ser (A,T,C,G). Por lo tanto, en caso que se envíe un carácter que no se encuentre en los aceptados, el programa lo reporta como excepción por carácter invalido.

Esta validación se ejecuta en la siguiente parte del código:



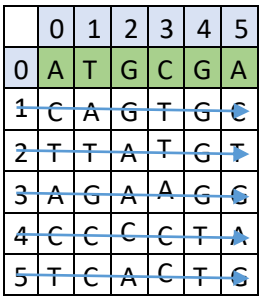
```
/**
 * Metodo principal que permite validar una matriz de datos para evaluar si
 * es correcta para su procesamiento Se realizan validaciones de estructura
 * de datos de la matriz, si tiene el tamaño minimo de comparacion, si es
 * cuadrada y si todos sus caracteres son validos
 *
 * @param datos
 */
private void validaDatosADN(String[] datos) {
    // Calculo el tamaño del parametro, si no es el adecuado se retorna error por datos insuficientes:
    if (datos.length < Integer.parseInt(DeteccionMutanteConstantesEnum.TAMANIO_MINIMO_COMPARACION.valor)) {
        log.error(ErroresValidacionEnum.MENSAJE_TAMANIO_MATRIZ_INVALIDO.mensaje);
        throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, ErroresValidacionEnum.MENSAJE_TAMANIO_MATRIZ_INVALIDO.mensaje);
    }
    List<String> datosList = Arrays.asList(datos);
    datosList.stream().map((dato) -> Arrays.asList(dato.split(""))).forEachOrdered((letras) -> {
        // Verifico si la matriz es cuadrada, si no es cuadrada se retorna error por estructura invalida
        if (Integer.compare(letras.size(), datos.length) != 0) {
            log.error(String.format(ErroresValidacionEnum.MENSAJE_NO_MATRIZ_CUADRADA.mensaje, datos.length, letras.size()));
            throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, String.format(ErroresValidacionEnum.MENSAJE_NO_MATRIZ_CUADRADA.mensaje, datos.length,
        }
        // Valido si los caracteres incluidos en cada posicion de la matriz estan dentro del rango de letras valido
        letras.forEach((letra) -> {
            if (!Arrays.asList(DeteccionMutanteConstantesEnum.CARACTERES_MATRIZ_VALIDOS.valor.split(DeteccionMutanteConstantesEnum.SEPARADOR_VALORES.valor)).contains(let
                log.error(String.format(ErroresValidacionEnum.MENSAJE_CARACTER_INVALIDO.mensaje, letra.toUpperCase()));
                throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, String.format(ErroresValidacionEnum.MENSAJE_CARACTER_INVALIDO.mensaje, letra.toUpp
            }
        });
    });
```

Donde cada carácter identificado por el atributo letra en el ciclo se valida ante un arreglo estático de letras con los caracteres aceptados:

```
*/
package co.com.ml.challenge.constantes;

/**
 *
 * @author usuario
 */
public enum DeteccionMutanteConstantesEnum {

    TAMANIO_MINIMO_COMPARACION("4"),
    CARACTERES_MATRIZ_VALIDOS("A,T,C,G"),
    SEPARADOR_VALORES(",");

    public final String valor;

    private DeteccionMutanteConstantesEnum(String valor) {
        this.valor = valor;
    }

}
```

Luego de esas superadas esas validaciones, se procede a ejecutar la validación de la cadena para identificar si el ADN corresponde a un mutante o a un humano. Para identificar el subconjunto de datos a evaluar para verificar si un carácter se repite se requiere recorrer de la siguiente forma, se inicia desde una posición:

Subconjunto Horizontal: Fila constante, columna aumenta en uno.

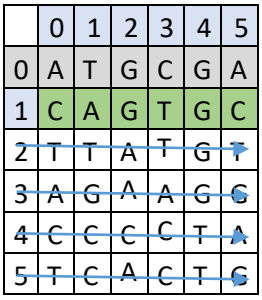


	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Para el caso gráfico, comenzando en la fila 0, se aumenta la columna en 1 hasta el final de la matriz para identificar el subconjunto a validar. Para el caso de la fila 0 el subconjunto a validar es el resaltado en verde:

Subconjunto=[A,T,G,C,G,A]

Luego de forma automática, se aumenta la fila para obtener el siguiente subconjunto.



	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Para el caso gráfico, comenzando en la fila 1 se aumenta la columna en 1 hasta el final de la matriz para identificar el subconjunto a validar. Para el caso de la fila 1 el subconjunto a validar es el resaltado en verde:

Subconjunto=[C,A,G,T,G,C]

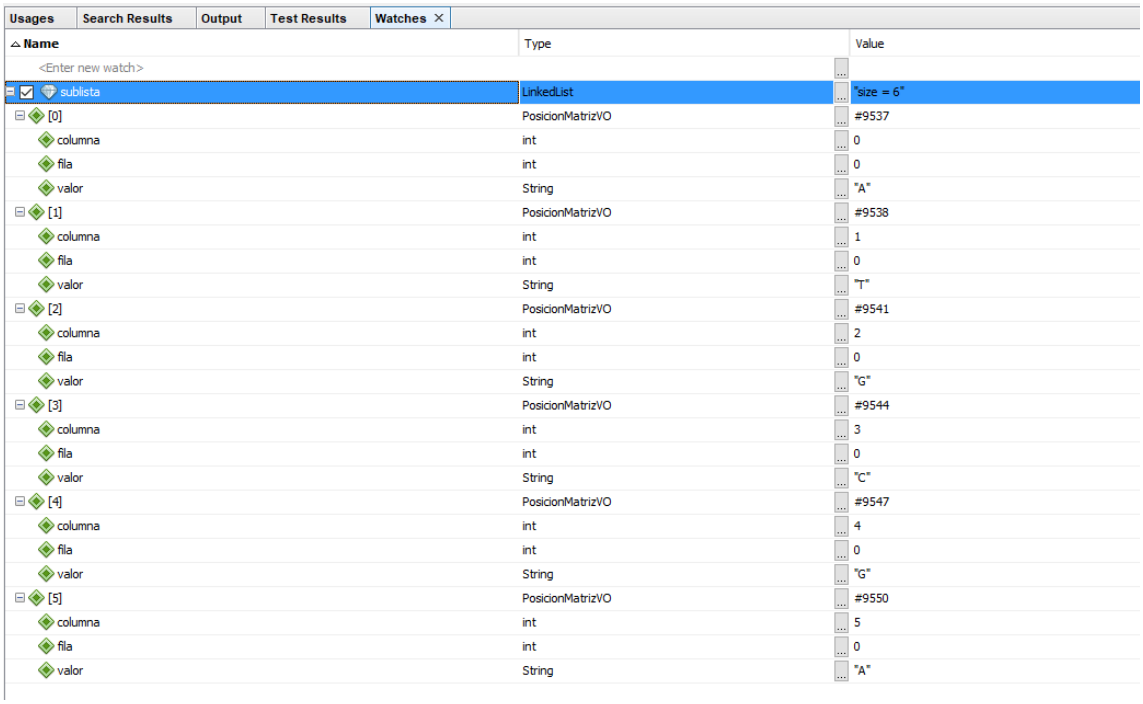
Esto se realiza en el siguiente método:

```
/**
 * Metodo que permite validar la recurrencia de un caracter en un
 * subconjunto de caracteres representados en una cadena de ADN. Los tipos
 * de validacion son, HORIZONTAL: De arriba hacia abajo, VERTICAL: De
 * izquierda a derecha y OBLICUO: Diagonales principales y secundarias. La
 * validacion se ejecuta desde un punto inicial y se calcula el subconjunto
 * desde esa cordenada especifica.
 *
 * @param matrizADN
 * @param fila
 * @param columna
 * @param caso
 * @return boolean TRUE en caso que sea mutante, FALSE en caso que sea
 * humano
 */
private boolean validacionMutante(List<PosicionMatrizVO> matrizADN, int fila, int columna, MetodoAgrupamientoEnum caso) {
    boolean retorno = false;
    // Se obtiene el subgrupo de datos para validación según el mecanismo (HORIZONTAL, VERTICAL U OBLICUO)
    List<PosicionMatrizVO> sublista = new LinkedList<>();
    if (MetodoAgrupamientoEnum.HORIZONTAL.equals(caso)) {
        sublista.addAll(matrizADN.stream().filter(prdct -> prdct.getFila() == fila).collect(toList()));
    } else if (MetodoAgrupamientoEnum.VERTICAL.equals(caso)) {
        sublista.addAll(matrizADN.stream().filter(prdct -> prdct.getColumna() == columna).collect(toList()));
    } else if (MetodoAgrupamientoEnum.GRUPO_OBLICUO.contains(caso)) {
        sublista.addAll(obtieneSubListaDiagonales(fila, columna, matrizADN, caso));
    }
```

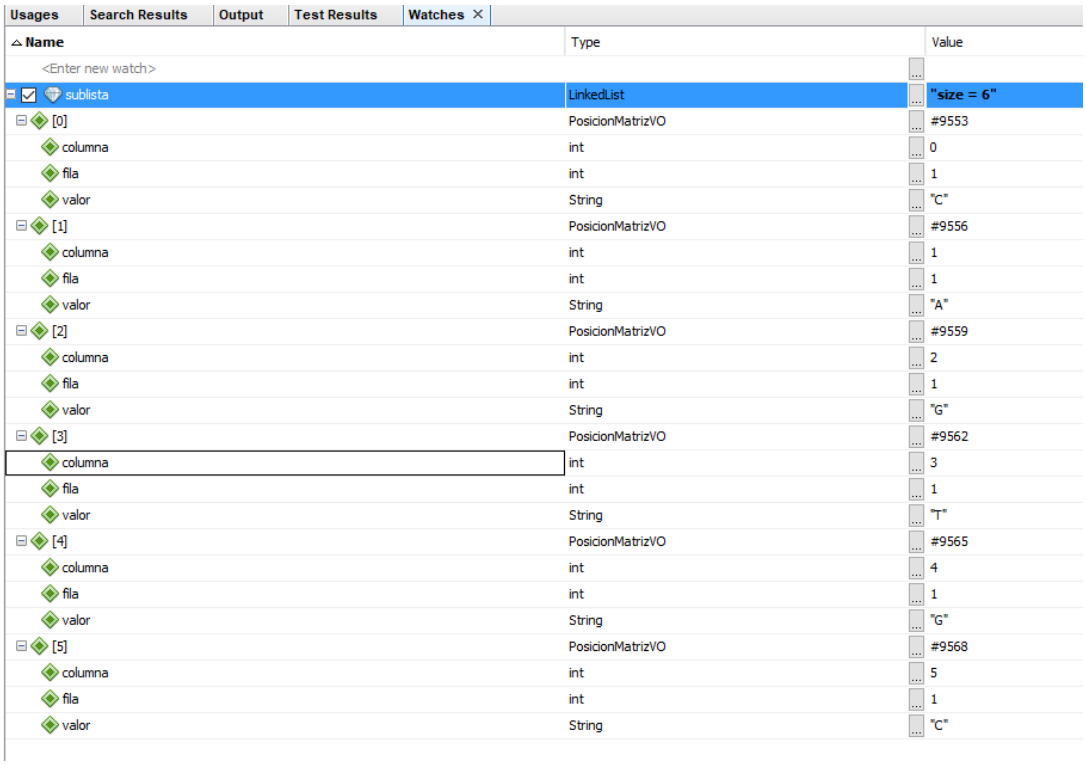
Como se ve, se selecciona el subconjunto de datos con la fila indicada, el resultado es una colección o sub-lista con los datos indicados en el algoritmo que mencione arriba.

La representación de los objetos en el programa son los siguientes:

Fila 0 - Subconjunto=[A,T,G,C,G,A]



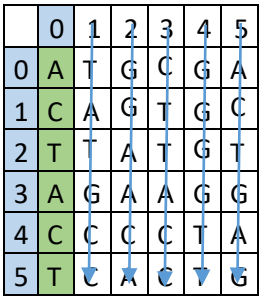
Fila 1 - Subconjunto=[C,A,G,T,G,C]



Usages	Search Results	Output	Test Results	Watches ✕

Name	Type	Value
<Enter new watch>		
sublista	LinkedList	"size = 6"
[0]	PosicionMatrizVO	#9553
columna	int	0
fila	int	1
valor	String	"C"
[1]	PosicionMatrizVO	#9556
columna	int	1
fila	int	1
valor	String	"A"
[2]	PosicionMatrizVO	#9559
columna	int	2
fila	int	1
valor	String	"G"
[3]	PosicionMatrizVO	#9562
columna	int	3
fila	int	1
valor	String	"T"
[4]	PosicionMatrizVO	#9565
columna	int	4
fila	int	1
valor	String	"G"
[5]	PosicionMatrizVO	#9568
columna	int	5
fila	int	1
valor	String	"C"

Subconjunto Vertical: Columna constante, fila aumenta en uno.

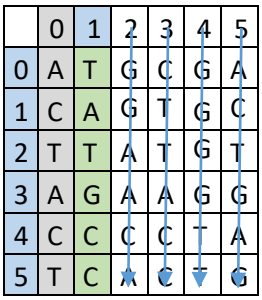


	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Para el caso gráfico, comenzando en la columna 0, se aumenta la fila en 1 hasta el final de la matriz para identificar el subconjunto a validar. Para el caso de la columna 0 el subconjunto a validar es el resaltado en verde:

Subconjunto=[A,C,T,A,C,T]

Luego de forma automática, se aumenta la columna para obtener el siguiente subconjunto.



	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Para el caso gráfico, comenzando en la columna 1 se aumenta la fila en 1 hasta el final de la matriz para identificar el subconjunto a validar. Para el caso de la columna 1 el subconjunto a validar es el resaltado en verde:

Subconjunto=[T,A,T,G,C,C]

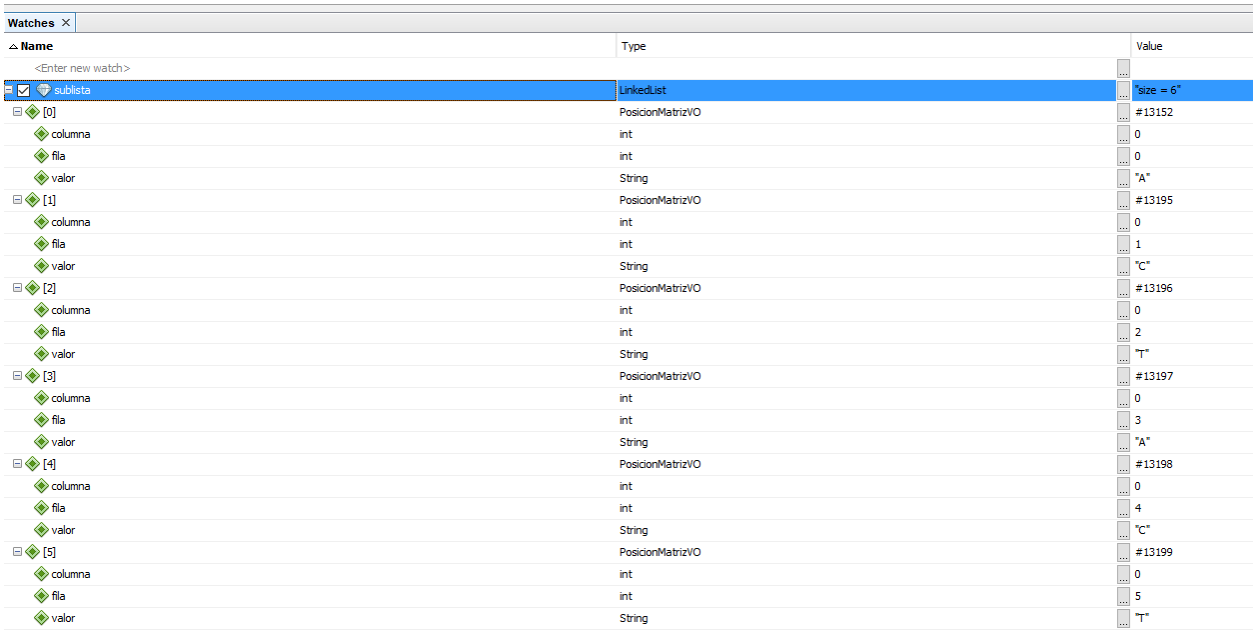
Esto se realiza en el siguiente método:

```
/**
 * Metodo que permite validar la recurrencia de un caracter en un
 * subconjunto de caracteres representados en una cadena de ADN. Los tipos
 * de validacion son, HORIZONTAL: De arriba hacia abajo, VERTICAL: De
 * izquierda a derecha y OBLICUO: Diagonales principales y secundarias. La
 * validacion se ejecuta desde un punto inicial y se calcula el subconjunto
 * desde esa cordenada especifica.
 *
 * @param matrizADN
 * @param fila
 * @param columna
 * @param caso
 * @return boolean TRUE en caso que sea mutante, FALSE en caso que sea
 * humano
 */
private boolean validacionMutante(List<PosicionMatrizVO> matrizADN, int fila, int columna, MetodoAgrupamientoEnum caso) {
    boolean retorno = false;
    // Se obtiene el subgrupo de datos para validación según el mecanismo (HORIZONTAL, VERTICAL U OBLICUO)
    List<PosicionMatrizVO> sublista = new LinkedList<>();
    if (MetodoAgrupamientoEnum.HORIZONTAL.equals(caso)) {
        sublista.addAll(matrizADN.stream().filter(prdct -> prdct.getFila() == fila).collect(toList()));
    } else if (MetodoAgrupamientoEnum.VERTICAL.equals(caso)) {
        sublista.addAll(matrizADN.stream().filter(prdct -> prdct.getColumna() == columna).collect(toList()));
    } else if (MetodoAgrupamientoEnum.GRUPO_OBLICUO.contains(caso)) {
        sublista.addAll(obtieneSubListaDiagonales(fila, columna, matrizADN, caso));
    }
```

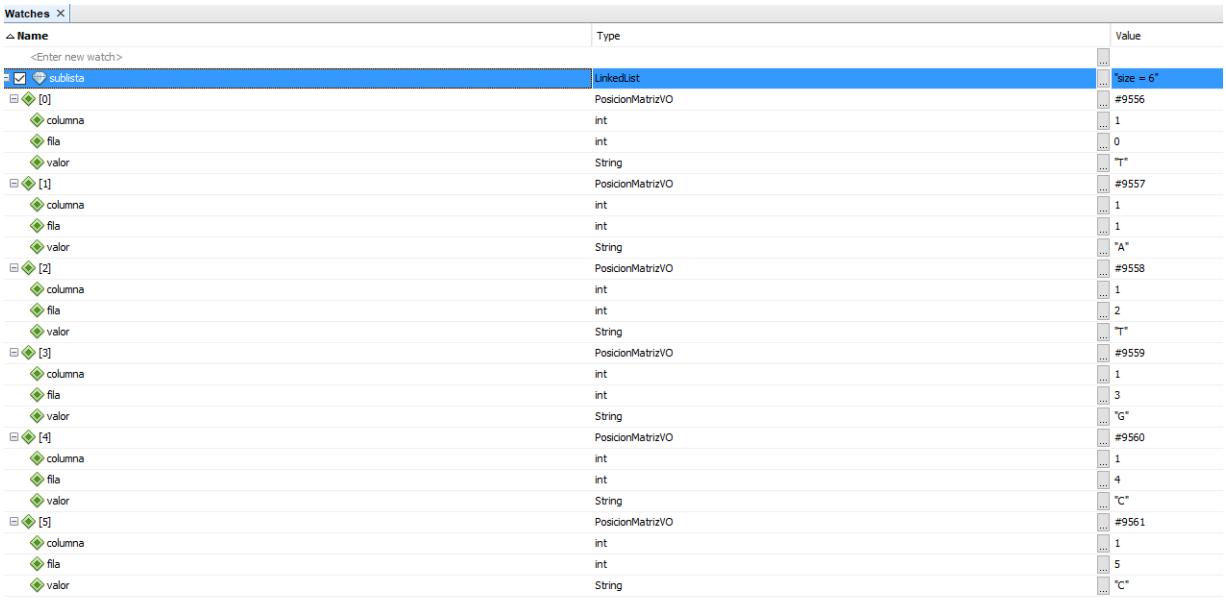
Como se ve, se selecciona el subconjunto de datos con la columna indicada, el resultado es una colección o sub-lista con los datos indicados en el algoritmo que mencione arriba.

La representación de los objetos en el programa son los siguientes:

Columna 0 - Subconjunto=[A,C,T,A,C,T]



Columna 1 - Subconjunto=[T,A,T,G,C,C]



Subconjunto Oblicuo: Fila y Columna con recorrido.

Para los datos oblicuos se efectúan dos distintos tipos de diagonal para la matriz cuadrada. Diagonal Principal y Diagonal Secundaria.

DIAGONAL PRINCIPAL

La diagonal principal corresponde a las de datos diagonales en recorrido desde la posición (0,0) hasta la posición (N,N). Esta diagonal cruza desde la parte superior izquierda hasta la parte inferior derecha. Para el caso de la matriz de ADN ejemplo, se identifica de la siguiente forma:

	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Para este caso se representa con la [fila=0] y [columna=0] y se suma una posición tanto a la fila como a la columna para encontrar la siguiente posición en la diagonal, hasta su límite final (cinco (5) en este escenario).

A partir de este punto se comienza a recorrer a partir de su índice (Sumatoria de una unidad por recorrido). Si se suma la fila, se recorre la diagonal inferior, mientras si se suma la columna, se recorre la diagonal superior. Gráficamente se representa así:

DIAGONAL PRINCIPAL INFERIOR

	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Caso [fila=1] y [columna=0], sumando una posición tanto a la fila como a la columna para encontrar la siguiente posición en la diagonal, hasta su límite final (cinco (5) en este escenario).

Se aumenta una posición del índice en fila hasta que se completa toda la diagonal. (Posición [fila=5] y [columna=0] con valor T).

DIAGONAL PRINCIPAL SUPERIOR

	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Caso [fila=0] y [columna=1], sumando una posición tanto a la fila como a la columna para encontrar la siguiente posición en la diagonal, hasta su límite final (cinco (5) en este escenario).

Se aumenta una posición del índice en fila hasta que se completa toda la diagonal. (Posición [fila=0] y [columna=5] con valor A).

Este algoritmo se ejecuta en la siguiente parte:

```
    }

    /**
     * Metodo que permite obtener los datos y posicion para ubicaciones oblicuas
     * dentro de la matriz de ADN. Segun el mecanismo de validacion se recorren
     * desde la diagonal principal (Desde arriba a la izquierda hasta abajo a la
     * derecha) o Diagonal secundaria (Desde arriba a la derecha hasta abajo a
     * la izquierda) y sus complementos superior e inferior.
     *
     * @param fila
     * @param columna
     * @param matriz
     * @param caso
     * @return List<PosicionMatrizVO> Colleccion de atributos correspondientes a
     * la diagonal evaluada.
     */
    private List<PosicionMatrizVO> obtieneSubListaDiagonales(int fila, int columna, List<PosicionMatrizVO> matriz, MetodoAgrupamientoEnum caso) {
        List<PosicionMatrizVO> sublista = new LinkedList<>();
        for (PosicionMatrizVO posicionMatrizVO : matriz) {
            if (posicionMatrizVO.getFila() == fila && posicionMatrizVO.getColumna() == columna) {
                sublista.add(posicionMatrizVO);
                if (caso.equals(MetodoAgrupamientoEnum.DIAGONAL_PRINCIPAL_INFERIOR) || caso.equals(MetodoAgrupamientoEnum.DIAGONAL_PRINCIPAL_SUPERIOR)) {
                    fila++;
                    columna++;
                } else if (caso.equals(MetodoAgrupamientoEnum.DIAGONAL_SECUNDARIA_INFERIOR) || caso.equals(MetodoAgrupamientoEnum.DIAGONAL_SECUNDARIA_SUPERIOR)) {
                    fila++;
                    columna--;
                }
            }
        }
        return sublista;
    }
```

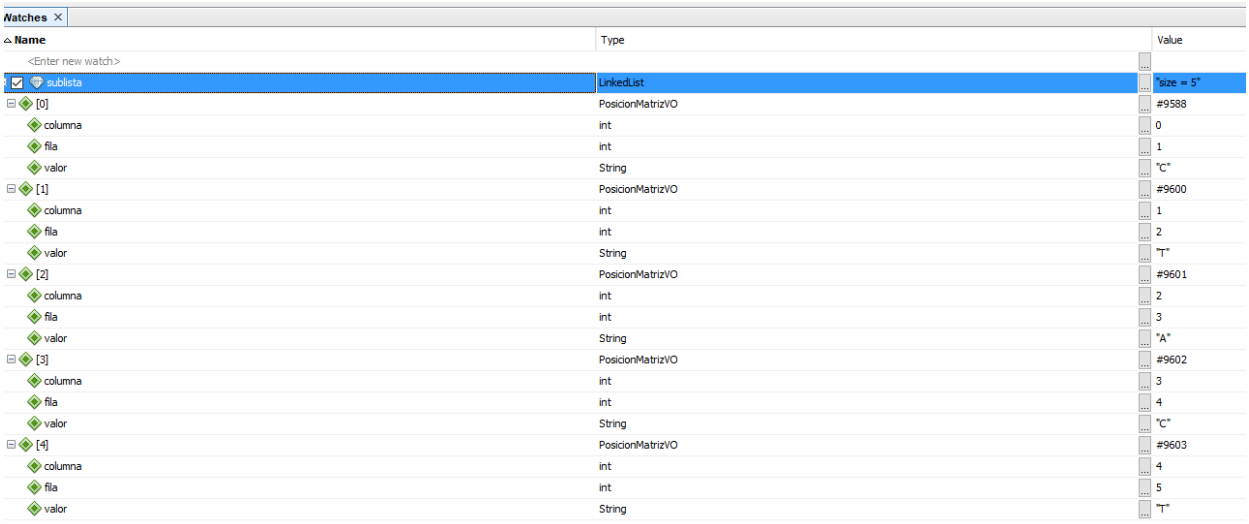
Donde se recibe el parámetro de la fila y o columna con el aumento de índice y se recorre en una posición la fila y la columna en toda la matriz.

Diagonal **principal**: (Desde Fila=0 – Columna=0 hasta Fila=5 – Columna=5) - Subconjunto=[A,A,A,A,T,G]

Watches

Name	Type	Value
<Enter new watch>		
sublista	LinkedList	"size = 6"
[0]	PosicionMatrizVO	#9553
columna	int	0
fila	int	0
valor	String	"A"
[1]	PosicionMatrizVO	#9554
columna	int	1
fila	int	1
valor	String	"A"
[2]	PosicionMatrizVO	#9555
columna	int	2
fila	int	2
valor	String	"A"
[3]	PosicionMatrizVO	#9556
columna	int	3
fila	int	3
valor	String	"A"
[4]	PosicionMatrizVO	#9557
columna	int	4
fila	int	4
valor	String	"T"
[5]	PosicionMatrizVO	#9558
columna	int	5
fila	int	5
valor	String	"G"

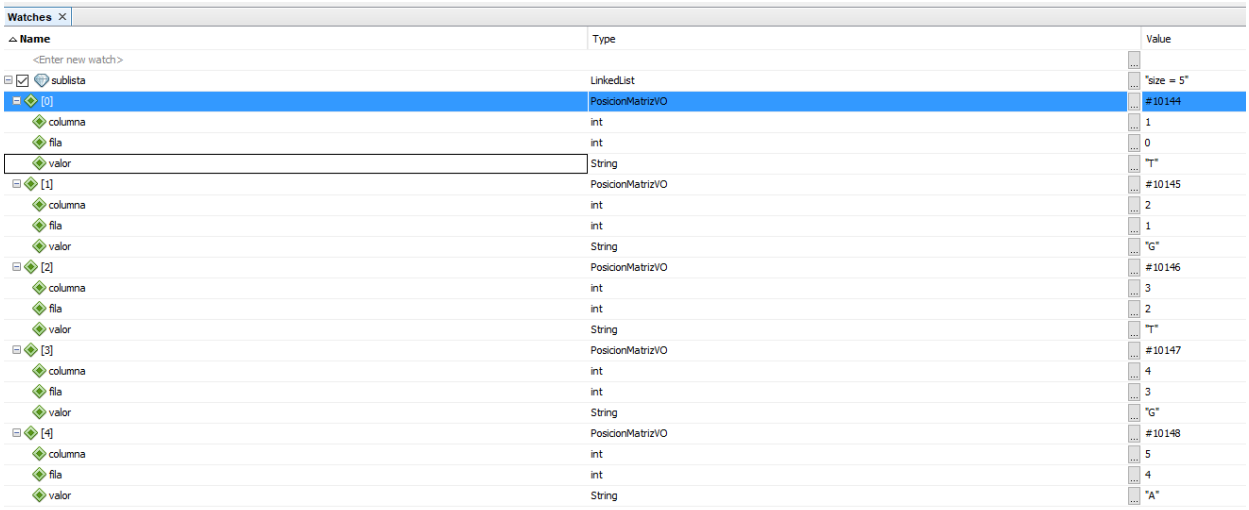
Diagonal principal **inferior**: (Desde Fila=1 – Columna=0 hasta Fila=5 – Columna=4) – Subconjunto =[C,T,A,C,T]



Name	Type	Value
<Enter new watch>		
sublista	LinkedList	"size = 5"
[0]	PosicionMatrizVO	#9588
columna	int	0
fila	int	1
valor	String	"C"
[1]	PosicionMatrizVO	#9600
columna	int	1
fila	int	2
valor	String	"T"
[2]	PosicionMatrizVO	#9601
columna	int	2
fila	int	3
valor	String	"A"
[3]	PosicionMatrizVO	#9602
columna	int	3
fila	int	4
valor	String	"C"
[4]	PosicionMatrizVO	#9603
columna	int	4
fila	int	5
valor	String	"T"

Nota: El recorrido se hace hasta el final del tamaño de la matriz.

Diagonal principal **superior**: (Desde Fila=0 - Columna=1 hasta Fila=4 - Columna=5) – Subconjunto =[T,G,T,G,A]



Name	Type	Value
<Enter new watch>		
sublista	LinkedList	"size = 5"
[0]	PosicionMatrizVO	#10144
columna	int	1
fila	int	0
valor	String	"T"
[1]	PosicionMatrizVO	#10145
columna	int	2
fila	int	1
valor	String	"G"
[2]	PosicionMatrizVO	#10146
columna	int	3
fila	int	2
valor	String	"T"
[3]	PosicionMatrizVO	#10147
columna	int	4
fila	int	3
valor	String	"G"
[4]	PosicionMatrizVO	#10148
columna	int	5
fila	int	4
valor	String	"A"

Nota: El recorrido se hace hasta el final del tamaño de la matriz.

DIAGONAL SECUNDARIA

La diagonal secundaria corresponde a las de datos diagonales en recorrido desde la posición (0,N) hasta la posición (N,0). Esta diagonal cruza desde la parte superior derecha hasta la parte inferior izquierda. Para el caso de la matriz de ADN ejemplo, se identifica de la siguiente forma:

	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Para este caso se representa con la [fila=0] y [columna=5] y se suma una posición a la fila, se resta una posición a la columna para encontrar la siguiente posición en la diagonal, hasta su límite final (cinco (5) en este escenario).

A partir de este punto se comienza a recorrer a partir de su índice (Sumatoria de una unidad por recorrido). Si se suma la fila, se recorre la diagonal inferior, mientras si se resta la columna, se recorre la diagonal superior. Gráficamente se representa así:

DIAGONAL SECUNDARIA INFERIOR

	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Caso [fila=1] y [columna=5], sumando una posición a la fila, restando una posición a la columna para encontrar la siguiente posición en la diagonal, hasta su límite final (cinco (5) en este escenario).

Se aumenta una posición del índice en columna hasta que se completa toda la diagonal. (Posición [fila=5] y [columna=5] con valor G).

Nota: El recorrido se hace hasta el final del tamaño de la matriz.

DIAGONAL SECUNDARIA SUPERIOR

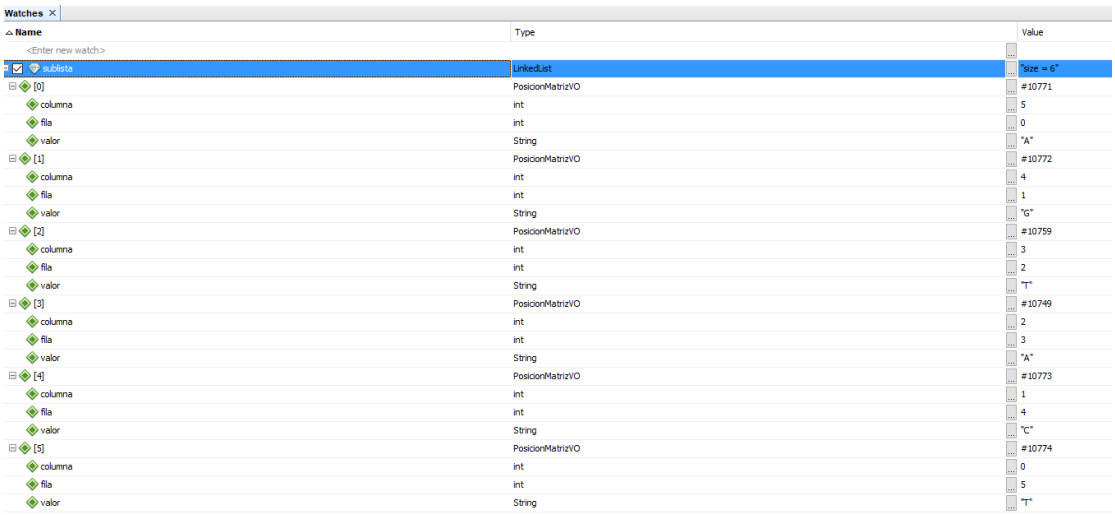
	0	1	2	3	4	5
0	A	T	G	C	G	A
1	C	A	G	T	G	C
2	T	T	A	T	G	T
3	A	G	A	A	G	G
4	C	C	C	C	T	A
5	T	C	A	C	T	G

Caso [fila=0] y [columna=4], sumando una posición a la fila, restando una posición a la columna para encontrar la siguiente posición en la diagonal, hasta su límite final (cinco (5) en este escenario).

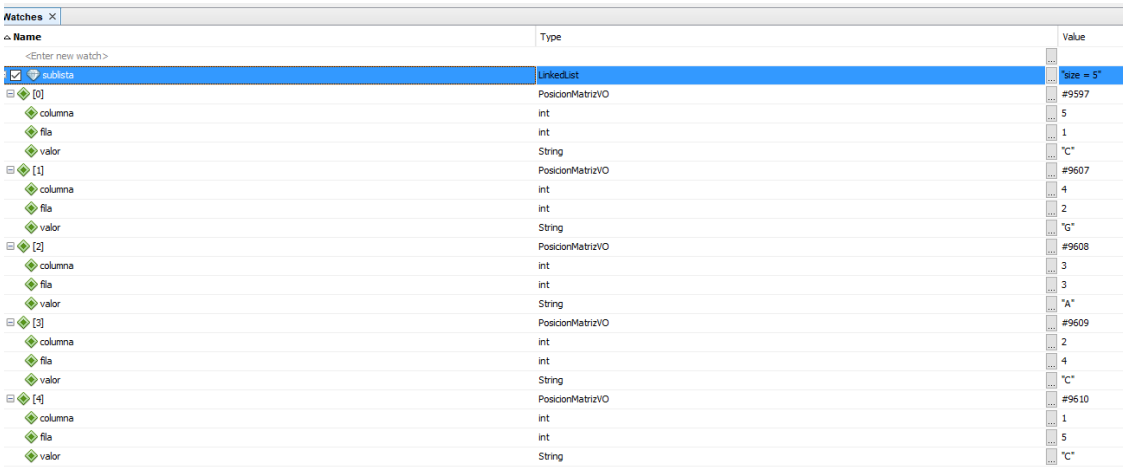
Se aumenta una posición del índice en fila hasta que se completa toda la diagonal. (Posición [fila=0] y [columna=0] con valor A).

Nota: El recorrido se hace hasta el final del tamaño de la matriz.

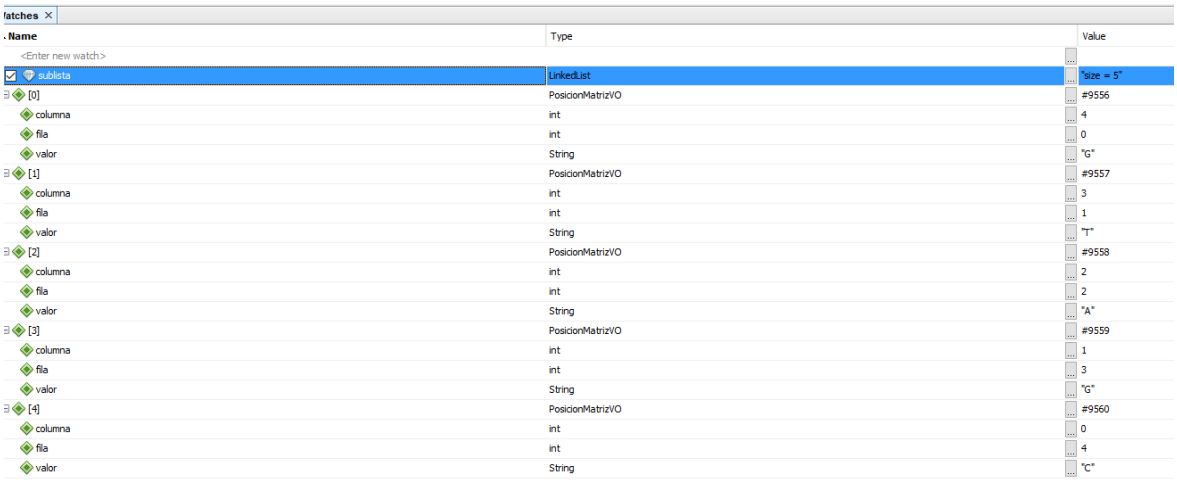
Diagonal **secundaria**: (Desde Fila=0 – Columna=5 hasta Fila=5 - Columna=0) - Subconjunto=[A,G,T,A,C,T]



Diagonal secundaria **inferior**: (Desde Fila=1 – Columna=5 hasta Fila=5 - Columna=1) - Subconjunto=[C,G,A,C,C]



Diagonal secundaria **superior**: (Desde Fila 0 - Columna 4 hasta Fila=0 - Columna=4) - Subconjunto=[G,T,A,G,C]



Una vez identificados los subconjuntos de datos (HORIZONTAL, VERTICAL Y OBLICUO), se procede a evaluar uno por uno si la recurrencia de un carácter se encuentra cuatro o más veces, en caso afirmativo el ADN se caracteriza como mutante, en caso contrario se caracteriza como humano.

Esto se realiza en la siguiente parte de código:

```
/* Si existen datos validos para comparar, se inicia la evaluacion de caracteres consecutivos */
if (!sublista.isEmpty()) {
    String valorInicial = sublista.get(0).getValor();
    int contadorRecurrencia = 0;
    for (PosicionMatrizVO posicionMatrizVO : sublista) {
        if (valorInicial.equals(posicionMatrizVO.getValor())) {
            contadorRecurrencia++;
        } else {
            /* Si es mayor o igual a 4, significa que es un mutante y rompe la iteracion de validacion */
            if (contadorRecurrencia >= Integer.parseInt(DeteccionMutanteConstantesEnum.TAMANIO_MINIMO_COMPARACION.valor)) {
                retorno = true;
                break;
            }
            valorInicial = posicionMatrizVO.getValor();
            contadorRecurrencia = 0;
        }
    }
}
return retorno;
```

La variable **valorInicial** recupera el primer registro del subconjunto y lo guarda para su comparación, en caso que durante el recorrido encuentre que el valor consecutivo es igual a la inicial, aumenta en uno el contador de recurrencia, en caso contrario, valida si el contador es igual o mayor a cuatro (4) y si lo es retorna TRUE, indicando que es un mutante. En este caso rompe el ciclo porque ya no tiene sentido seguir validando los siguientes subconjuntos. En caso que el contador de recurrencia no sea mayor o igual a cuatro (4), reemplaza el **valorInicial** por el siguiente carácter encontrado e inicializa el contador de recurrencia para validar el siguiente carácter. En caso que finalice el recorrido y no encuentre la recurrencia de cuatro (4) o más caracteres repetidos, se define que el ADN es humano y retorna FALSE.

Con esta validación se termina el **Nivel 1** del Challenge de Mercado Libre.

Nivel 2:

Para el nivel 2 se requiere la publicación del API en una nube publica, exponiendo un servicio de tipo REST mediante HTTP POST.

Por lo tanto, en este punto es requerido implementar un controlador que permita exportar estas Apis de forma pública. El controlador es el siguiente:

```
/**
 *
 * @author usuario
 */
@RestController
@Api(tags = "DeteccionMutante")
public class DeteccionMutanteController {

    @Autowired
    private DeteccionMutanteBusiness deteccionMutanteBusiness;

    @PostMapping(value = "/mutant", produces = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_JSON_VALUE})
    @ApiOperation(value = "Validacion de mutante segun su cadena de ADN", response = void.class, httpMethod = "POST")
    @ApiResponses({@ApiResponse(code = 200, message = "La cadena corresponde a la de un mutante"),
                   @ApiResponse(code = 403, message = "La cadena corresponde a la de un simple humano", response = DeteccionMutantesException.class),
                   @ApiResponse(code = 500, message = "Error de datos en la validacion", response = DeteccionMutantesException.class)})
    public void validarMutante(@Valid @RequestBody(required = true) DnaVO dna) {
        if (!deteccionMutanteBusiness.isMutant(dna.getDna())) {
            throw new DeteccionMutantesException(HttpStatus.FORBIDDEN, "No es mutante, es un simple humano");
        }
    }
```

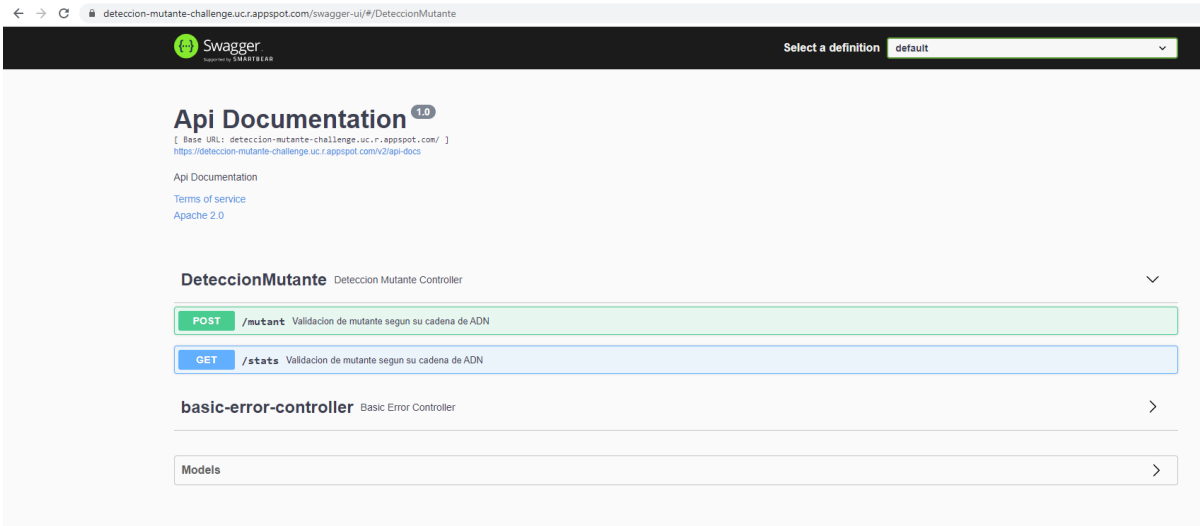
Este API tiene la estructura indicada por la solicitud. El valor del POST mapping es mutant y recibe como request body un JSON de tipo DnaVO, el cual es una representación del arreglo de strings que se planteó como parámetro:

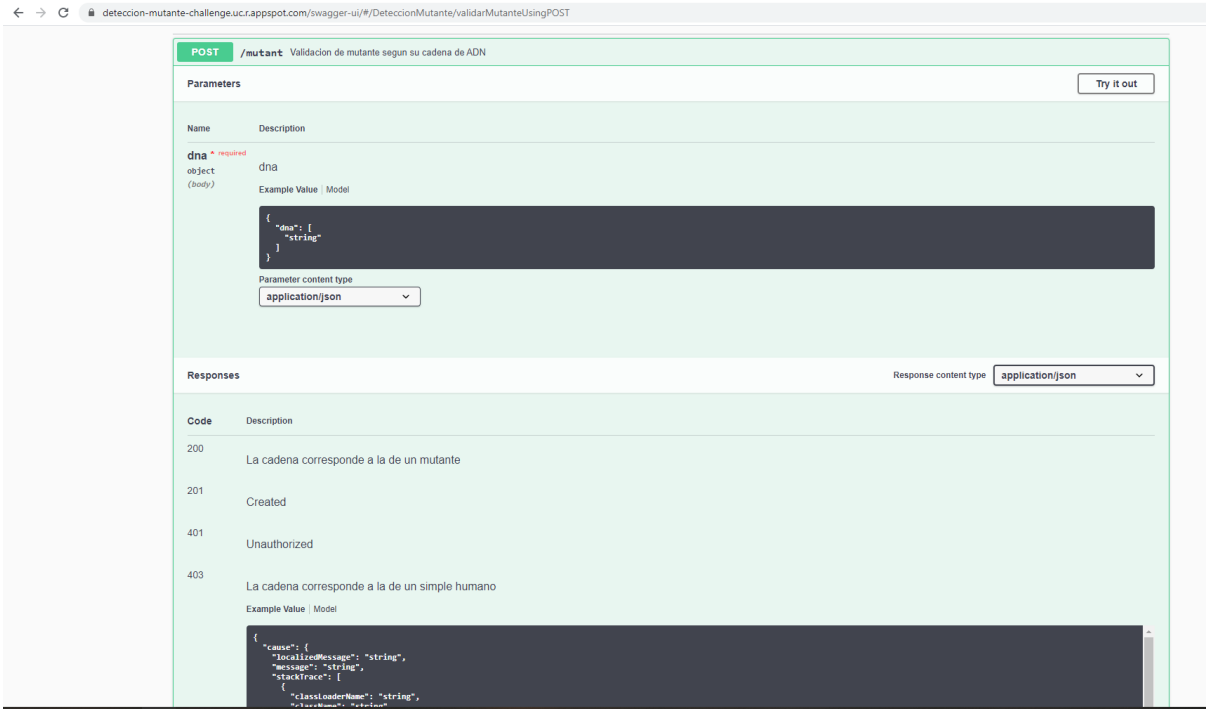
```java
package co.com.ml.challenge.vo;

import io.swagger.annotations.ApiModel;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

/**
 *
 * @author usuario
 */
@ApiModel
@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class DnaVO {

    String[] dna;
}
```

Como se indica en la implementación, se realiza una documentación de la firma con Swagger2, lo que permite visualizar de forma nativa la API, su consumo y sus respuestas a partir de una URL implícita en el microservicio.

La URL del Swagger para el API /mutant es el siguiente:

https://deteccion-mutante-challenge.uc.r.appspot.com/swagger-ui/#/

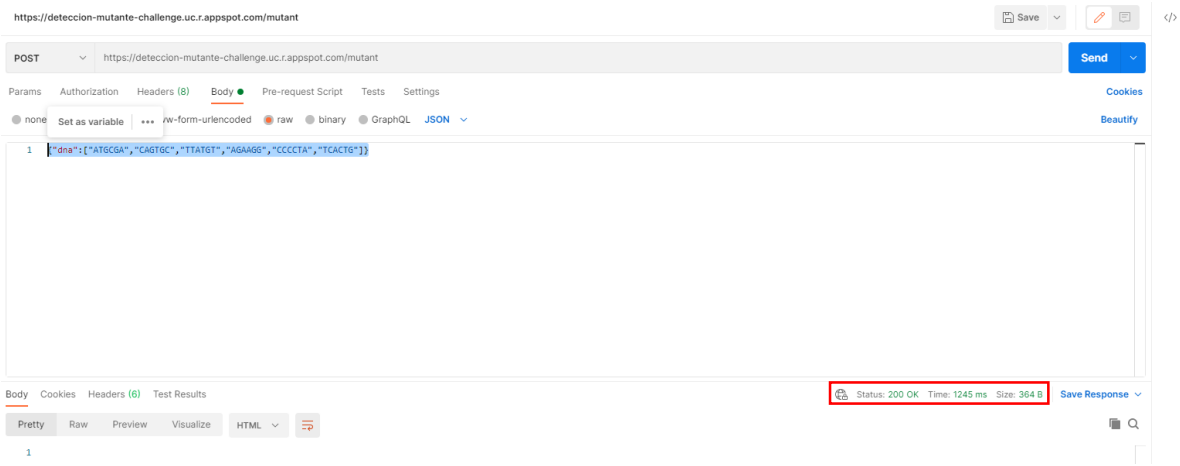




Al consumir el API, si la cadena corresponde a un mutante, debe responder un **200 OK**. Para este caso de prueba, se usa el mismo arreglo que se envía en el ejemplo para un **mutante**:

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

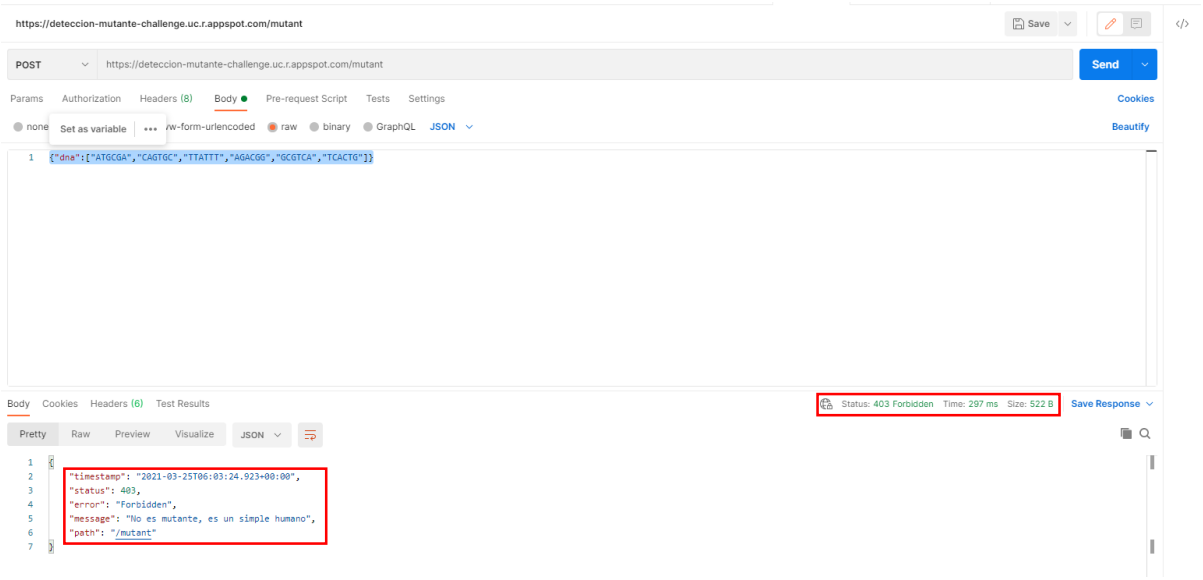
Para este caso, se evidencia que la respuesta es la solicitada en el formato requerido:



Para el caso que la cadena no corresponda a un mutante, debe responder un **403 Forbidden**. Para este escenario, se usa el mismo arreglo que se envía de ejemplo para un **humano**:

```
{"dna":["ATGCGA","CAGTGC","TTATTT","AGACGG","GCGTCA","TCACTG"]}
```

Para este caso, se evidencia que la respuesta es la solicitada en el formato requerido. Adicional, al ser una excepción se adiciona un mensaje descriptivo.



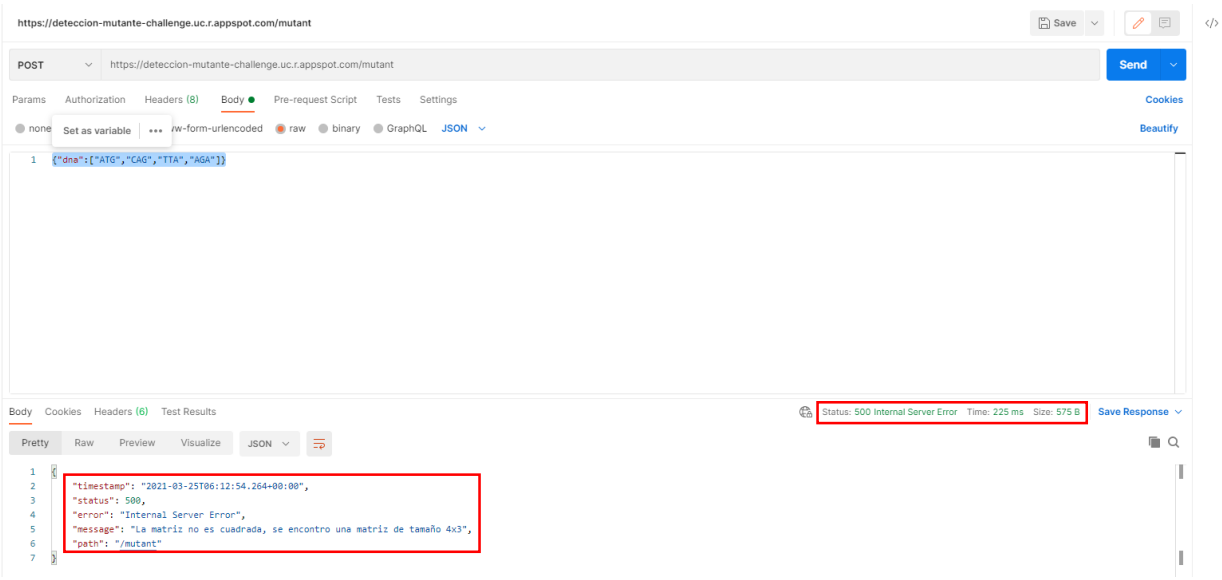
Adicional, se agregan una serie de excepciones de tipo **500 Internal Server Error** las cuales identifican los errores de **validación de estructura, datos inválidos y caracteres inválidos** que se ejecutan antes de la evaluación de una cadena de ADN.

Para la excepción de validación de estructura, utilizamos una cadena que no contenga el tamaño adecuado:

```
{"dna":["ATG","CAG","TTA","AGA"]}
```

Este parámetro comprende que la matriz no es cuadrada, ya que su formación final daría una dimensión de [4x3]. Al invocar el API este responde con el error 500.

	1	2	3
1	A	T	G
2	C	A	G
3	T	T	A
4	A	G	A

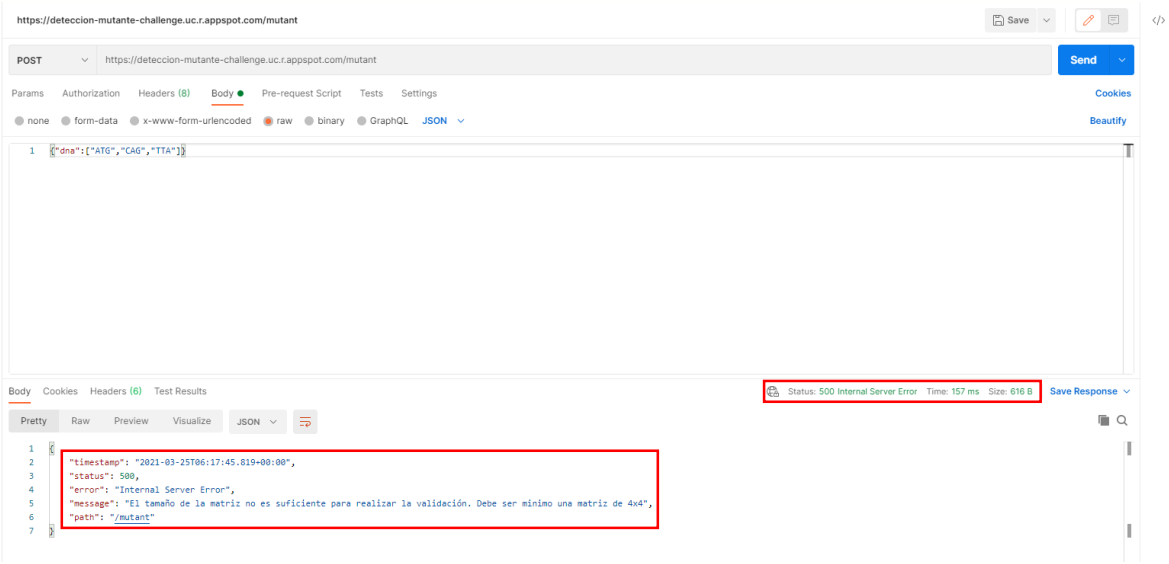


Para la excepción de datos inválidos, utilizamos una cadena que no contenga el tamaño mínimo para su validación, en el caso de la solicitud la cantidad mínima de repetición de caracteres consecutivos es cuatro (4):

```
{"dna":["ATG","CAG","TTA"]}
```

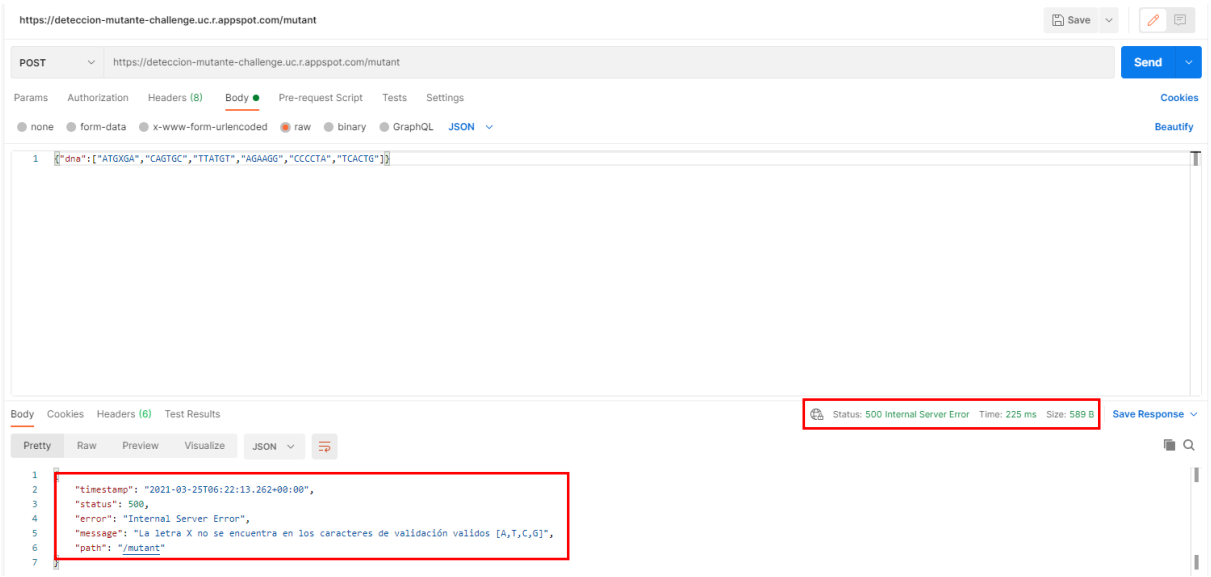
	1	2	3
1	A	T	G
2	C	A	G
3	T	T	A

A pesar que la matriz es cuadrada, los datos son insuficientes para realizar la evaluación. En este punto también se podría indicar que es humano, pero al no estar explícito en la solicitud se interpreta como excepción.



Finalmente, para el caso de caracteres inválidos, utilizamos una cadena con la estructura adecuada y los datos suficientes para la validación, pero incluyendo una X que no esta en la referencia de letras habilitadas para la cadena de ADN (A,T,C,G).

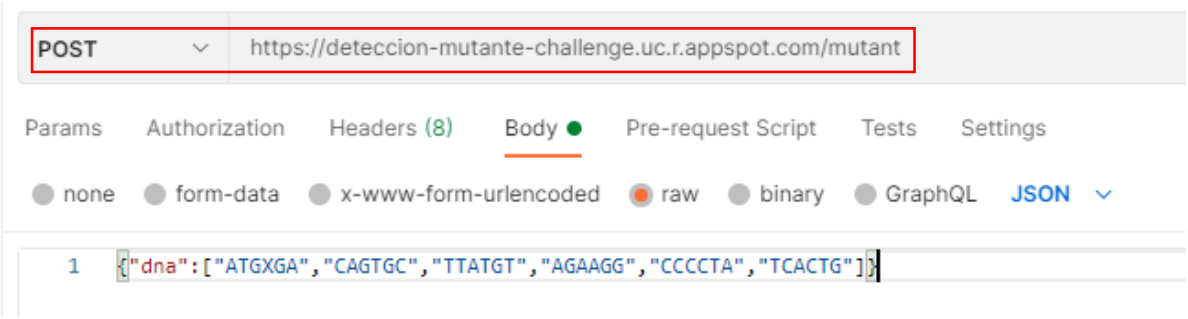
```
{"dna":["ATGXGA","CAGTGC","TTATGT","AGAAGG","CCCCTA","TCACTG"]}
```



La URL de consumo del API de validación de mutantes es la siguiente:

https://deteccion-mutante-challenge.uc.r.appspot.com/mutant

Su tipo es **POST.**



Con esta validación se termina el **Nivel 2** del Challenge de Mercado Libre.

Nivel 3:

Se realiza la solicitud de generar un API con estadísticas a partir de su persistencia en base de datos. Se requiere guardar un registro único por cadena de ADN y su caracterización según su validación.

Para este requerimiento, se crea un memory repository a partir de un repositorio de spring-data (JPARepository) llamado EstadisticasValidacionRepository con los métodos de inserción y consulta de datos necesarios para salvar la información de las peticiones necesarias.

Los métodos implementados son los siguientes:

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package co.com.ml.challenge.repository;

import co.com.ml.challenge.dao.EstadisticasValidacion;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;

/**
 *
 * @author usuario
 */
public interface EstadisticasValidacionRepository extends JpaRepository<EstadisticasValidacion, Long> {

    public EstadisticasValidacion findByCadenaAdn(@Param("cadenaAdn") String cadenaAdn);

    public Integer findHumanoCount();

    public Integer findMutanteCount();

}
```

public EstadisticasValidacion findByCadenaAdn(@Param “cadenaAdn” cadenaAdn)

Permite identificar si una cadena de ADN se encuentra ya evaluada por el sistema, en caso que no exista se persiste.

Esto se representa en el siguiente NamedQuery:

```java
@NamedQuery(name = "EstadisticasValidacion.findAll", query = "SELECT E FROM EstadisticasValidacion E")
@NamedQuery(name = "EstadisticasValidacion.findByCadenaAdn", query = "SELECT E FROM EstadisticasValidacion E where E.cadenaAdn = :cadenaAdn")
@NamedQuery(name = "EstadisticasValidacion.findHumanoCount", query = "SELECT COUNT(E) FROM EstadisticasValidacion E where E.esMutante = false")
@NamedQuery(name = "EstadisticasValidacion.findMutanteCount", query = "SELECT COUNT(E) FROM EstadisticasValidacion E where E.esMutante = true")
```

public Integer findHumanoCount();

Permite identificar la cantidad de humanos a partir de la evaluación de los registros que tengan el atributo esMutante en false.

Esto se representa en el siguiente NamedQuery:

```java
@NamedQuery(name = "EstadisticasValidacion.findAll", query = "SELECT E FROM EstadisticasValidacion E")
@NamedQuery(name = "EstadisticasValidacion.findByCadenaAdn", query = "SELECT E FROM EstadisticasValidacion E where E.cadenaAdn = :cadenaAdn")
@NamedQuery(name = "EstadisticasValidacion.findHumanoCount", query = "SELECT COUNT(E) FROM EstadisticasValidacion E where E.esMutante = false")
@NamedQuery(name = "EstadisticasValidacion.findMutanteCount", query = "SELECT COUNT(E) FROM EstadisticasValidacion E where E.esMutante = true")
```

public Integer findMutanteCount();

Permite identificar la cantidad de mutantes a partir de la evaluación de los registros que tengan el atributo esMutante en true.

Esto se representa en el siguiente NamedQuery:

```java
@NamedQuery(name = "EstadisticasValidacion.findAll", query = "SELECT E FROM EstadisticasValidacion E")
@NamedQuery(name = "EstadisticasValidacion.findByCadenaAdn", query = "SELECT E FROM EstadisticasValidacion E where E.cadenaAdn = :cadenaAdn")
@NamedQuery(name = "EstadisticasValidacion.findHumanoCount", query = "SELECT COUNT(E) FROM EstadisticasValidacion E where E.esMutante = false")
@NamedQuery(name = "EstadisticasValidacion.findMutanteCount", query = "SELECT COUNT(E) FROM EstadisticasValidacion E where E.esMutante = true")
```

La representación final de la entidad es la siguiente:

```java
/**
 *
 * @author usuario
 */
@Data
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "EstadisticasValidacion")
@NamedQuery(name = "EstadisticasValidacion.findAll", query = "SELECT E FROM EstadisticasValidacion E")
@NamedQuery(name = "EstadisticasValidacion.findByCadenaAdn", query = "SELECT E FROM EstadisticasValidacion E where E.cadenaAdn = :cadenaAdn")
@NamedQuery(name = "EstadisticasValidacion.findHumanoCount", query = "SELECT COUNT(E) FROM EstadisticasValidacion E where E.esMutante = false")
@NamedQuery(name = "EstadisticasValidacion.findMutanteCount", query = "SELECT COUNT(E) FROM EstadisticasValidacion E where E.esMutante = true")
public class EstadisticasValidacion implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "id", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "cadena_adn", nullable = false)
    private String cadenaAdn;

    @Column(name = "es_mutante", nullable = false)
    private Boolean esMutante;
```

Donde:

id: Corresponde a la primary key autoincremental de la entidad en un atributo de tipo long.

cadenaAdn: Corresponde a la representación del arreglo de la cadena de ADN en un atributo tipo String.

esMutante: Corresponde al atributo booleano que indica si esa cadena corresponde a un mutante o a un humano.

Al habilitar estos métodos, es posible realizar la implementación correspondiente para generar el servicio de estadísticas requerido.

```java
/**
 * Metodo que permite obtener los stats de validacion de mutantes y humanos,
 * asi como el ratio de mutantes
 *
 * @return EstadisticasVO Objeto que tiene la estructura de retorn con
 * cantidad de mutantes, humanos y ratio
 */
@Override
public EstadisticasVO obtieneEstadisticas() {
    Integer contadorHumanos = estadisticasValidacionRepository.findHumanoCount();
    Integer contadorMutantes = estadisticasValidacionRepository.findMutanteCount();
    Double ratio = new Double(contadorMutantes) / new Double(contadorHumanos);
    log.info("Cantidad de humanos: {}, Cantidad de mutantes: {}, Ratio: {}", contadorHumanos, contadorMutantes, ratio);
    EstadisticasVO estadisticasVO = new EstadisticasVO(contadorMutantes, contadorHumanos, ratio);
    return estadisticasVO;
}
```

Inicialmente se contabiliza el total de humanos registrados hasta el momento de la petición, luego se contabiliza el total de mutantes registrados hasta el momento de la petición y finalmente se operan para generar la ratio, a partir de la división de mutantes vs humanos. Recordemos que cada registro en base de datos tiene una caracterización por lo que al sumar los datos de las dos consultas nos da el total de validaciones **únicas** que se han efectuado en el sistema.

Finalmente, se setean los atributos en el objeto EstadisticasVO que tiene la representación de JSON que se indica en el response del API.

Para poblar los datos de esta entidad, se ejecuta la persistencia en el momento que se realiza la validación de una cadena de ADN. Esto se representa en el siguiente código:

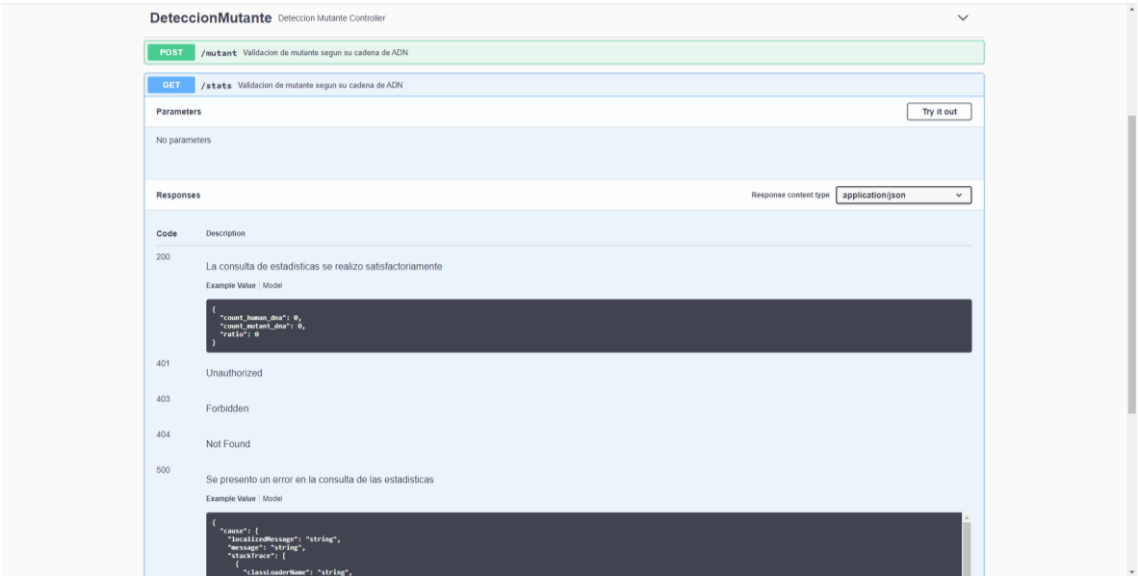
```java
/**
 * Metodo que permite insertar un registro de validacion unico sea para
 * mutante como para humano.ç En caso que ya se encuentre en el repositorio,
 * se omite la inserciòn
 *
 * @param cadenaADN: Cadena de ADN que se valido en formato original
 * @param resultado: Booleano con el resultado de la validacion. TRUE en
 * caso que sea mutante, FALSE en caso que sea humano
 */
private void insertaRegistroEstadistico(String cadenaADN, Boolean resultado) {
    EstadisticasValidacion estadisticasValidacion = new EstadisticasValidacion();
    estadisticasValidacion.setCadenaAdn(cadenaADN);
    estadisticasValidacion.setEsMutante(resultado);
    if (estadisticasValidacionRepository.findByCadenaAdn(cadenaADN) == null) {
        estadisticasValidacionRepository.saveAndFlush(estadisticasValidacion);
        log.info("Se guarda la cadena de ADN {} en el repositorio con resultado {}", cadenaADN, resultado);
    } else {
        log.info("La cadena de ADN {} ya se encuentra en el repositorio", cadenaADN);
    }
}
```

Antes de la inserción, se evalúa si una cadena ya existe en el repositorio, en caso positivo se informa en el log de la aplicación, en caso contrario se realiza la persistencia en el repositorio.

Finalmente, se genera el API en el controlador para publicar la interfaz de acceso al servicio.

```java
@GetMapping(value = "/stats", produces = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_JSON_VALUE})
@ApiOperation(value = "Validacion de mutante segun su cadena de ADN", response = EstadisticasVO.class, httpMethod = "GET")
@ApiResponses({@ApiResponse(code = 200, message = "La consulta de estadisticas se realizo satisfactoriamente"),
        @ApiResponse(code = 500, message = "Se presento un error en la consulta de las estadisticas", response = DeteccionMutantesException.class)})
public EstadisticasVO consultarEstadisticas() {
    return deteccionMutanteBusiness.obtieneEstadisticas();
}
}
```

Como se identifica, se genera un método HTTP GET con el API /stats

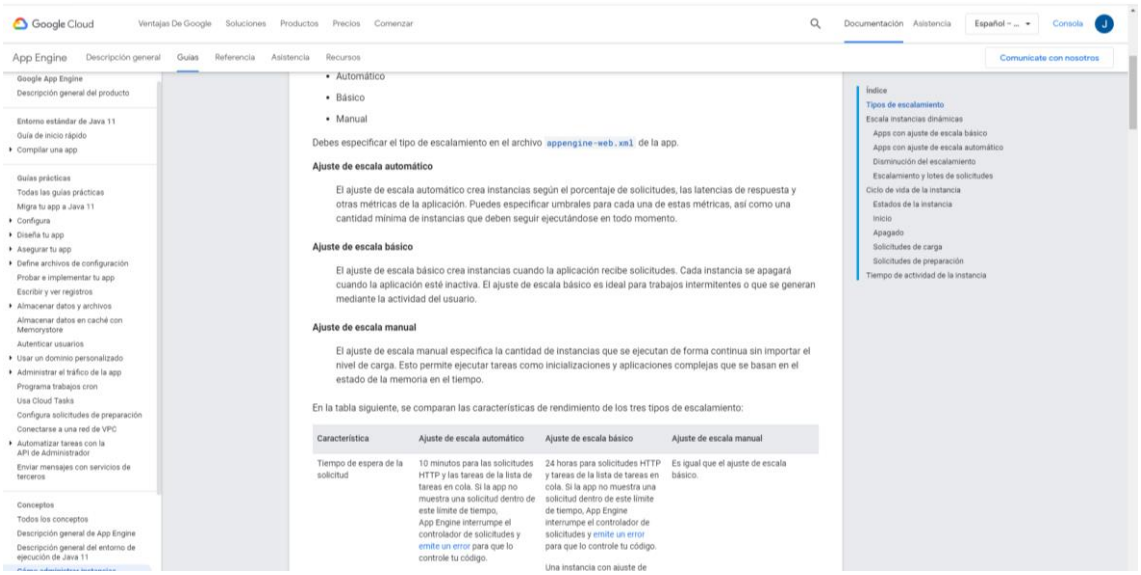


El swagger documental del API se encuentra en la ruta:

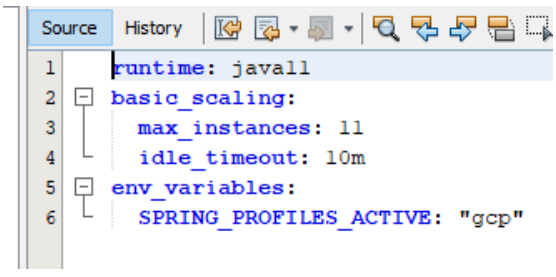
https://deteccion-mutante-challenge.uc.r.appspot.com/swagger-ui/#/

Se indica que el API puede tener peticiones agresivas en un periodo corto de tiempo. Al seleccionar Google App Engine, se identifica en la documentación que se trata de un entorno de trabajo tipo **B2**. Según su documentación y para ser ampliada por favor remitirse a:

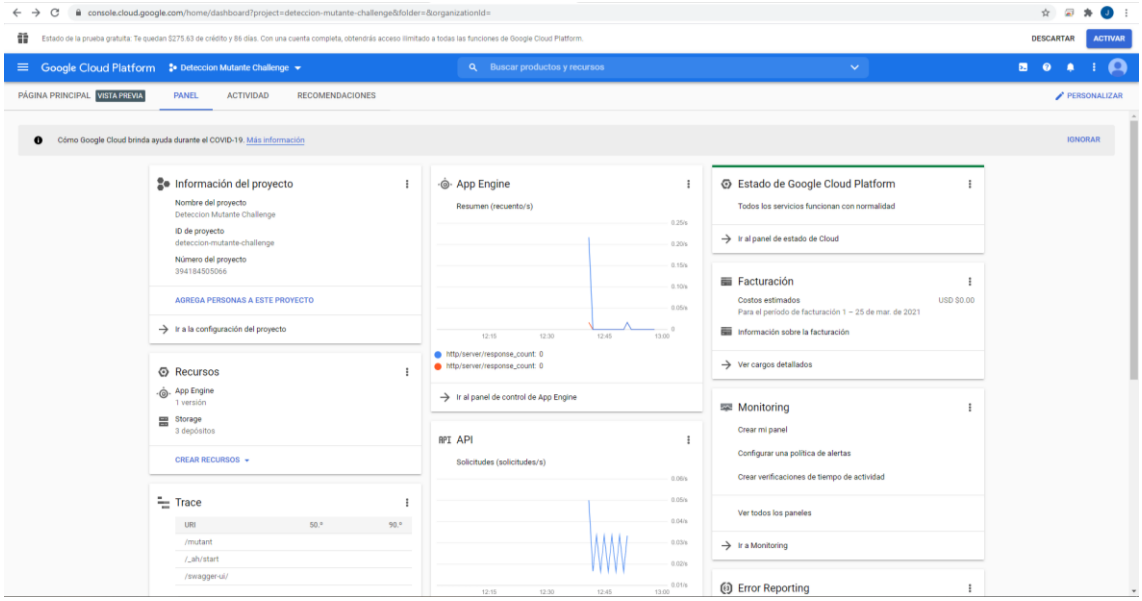
- https://cloud.google.com/appengine/docs/standard/java11/config/appref
- https://cloud.google.com/appengine/docs/standard/java11/how-instances-are-managed#scaling_types
- https://cloud.google.com/appengine/docs/standard#instance_classes



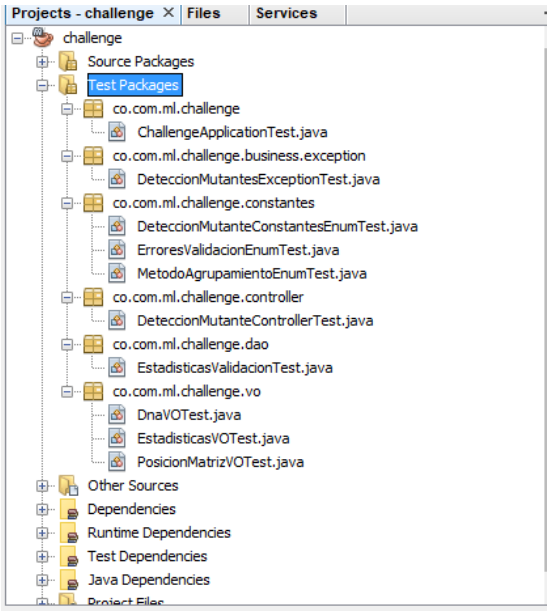
Se configuran los parámetros de aplicación con un escalamiento básico del entorno para soportar un rango de 11 instancias en 10m. Esto permite que el gestor de ambiente genere tantas instancias como sean posibles para mantener la disponibilidad del ambiente, impactando el tiempo de respuesta, ya que recrea las instancias desde cero. Esta definición inicial se selecciona por costos, pero en caso de una necesidad mayor se puede plantear la migración a una configuración de escalamiento automático y/o entorno flexible, configurando esta propiedad tanto en el balanceador de carga de los ambientes en nube como en el archivo de propiedades app.yaml



Con esto se realiza el despliegue en la nube de forma correcta.



Finalmente, se solicita una cobertura de pruebas automáticas mayor o igual a un 80% dentro de toda la implementación. Para ellos se realizan test cases automatizados a partir de JUnit basados en SpringTestCases. Se implementa una clase de pruebas por cada una de las clases de negocio desarrolladas.



Estas clases permiten cubrir la mayoría de métodos de la aplicación (Excepto algunos patrones builder implícitos en la librería lombok, utilizada para los atributos POJO). Para la ejecución de los contextos web se utiliza MockMvc, para las instancias de negocio se usa Mockito.

```java
/**
 *
 * @author usuario
 */
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class DeteccionMutanteControllerTest {

    @InjectMocks
    private DeteccionMutanteController deteccionMutanteController;

    @Mock
    private DeteccionMutanteBusiness deteccionMutanteBusiness;

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testValidarHumano() throws Exception {
        String requestJson = "{\"dna\":[\"ATGCGA\",\"CAGTGC\",\"TTATTT\",\"AGACGG\",\"GCGCCA\",\"TCACCG\"]}";
        mockMvc.perform(post("/mutant").contentType(MediaType.APPLICATION_JSON_VALUE).content(requestJson)).andExpect(status().isForbidden());
    }

    @Test
    public void testValidarMutante() throws Exception {
        String requestJson = "{\"dna\":[\"ATGCGA\",\"CAGTGC\",\"TTATGT\",\"AGAAGG\",\"CCCCTA\",\"TCACTG\"]}";
        mockMvc.perform(post("/mutant").contentType(MediaType.APPLICATION_JSON_VALUE).content(requestJson)).andExpect(status().isOk());
    }

    @Test
    public void obtieneEstadisticas() throws Exception {
        mockMvc.perform(get("/stats").contentType(MediaType.APPLICATION_JSON_VALUE)).andExpect(status().isOk());
```

```java
@Test
public void testConsultarEstadisticas() {
    EstadisticasVO estadisticasVO = new EstadisticasVO(0, 0, Double.NaN);
    Mockito.when(deteccionMutanteBusiness.obtieneEstadisticas()).thenReturn(estadisticasVO);
    EstadisticasVO respuestaListaAgenteDTO = deteccionMutanteController.consultarEstadisticas();
    assertTrue(estadisticasVO.equals(respuestaListaAgenteDTO));
}
```

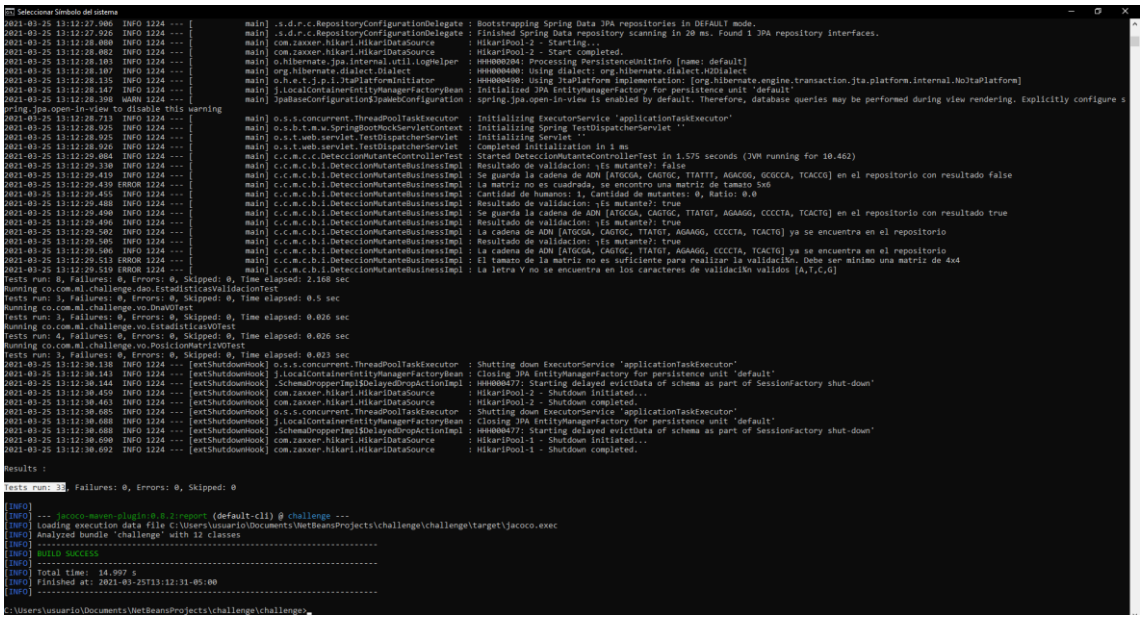
La ejecución se realiza desde el plugin de jacoco y su ejecución se configura a partir del surefile plugin de maven.

```xml
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.2</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>report</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.11</version>
    <configuration>
        <includes>
            <include>**/*Test.java</include>
        </includes>
    </configuration>
</plugin>
```

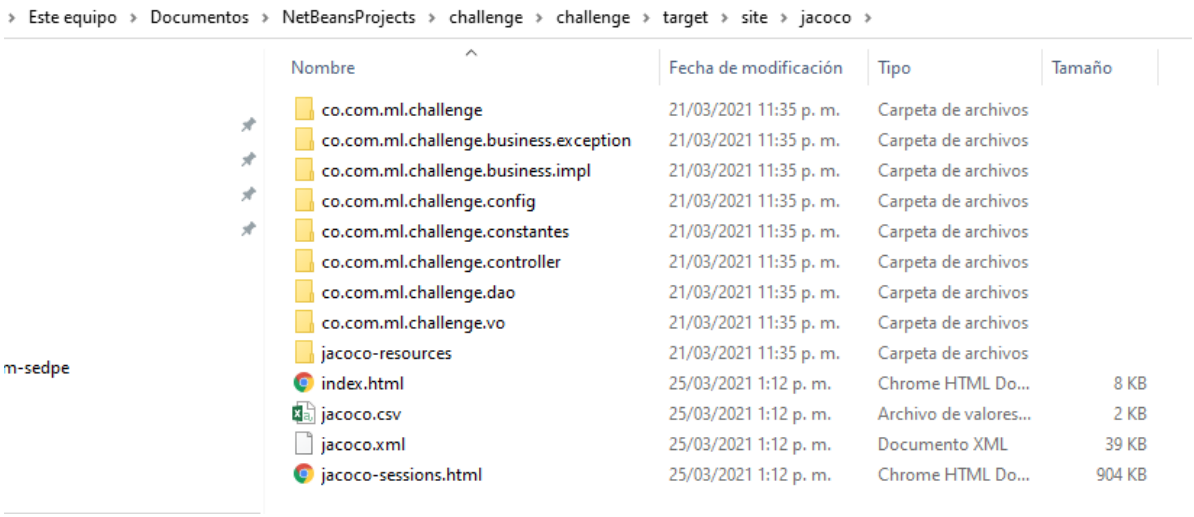
Esto garantiza que en tiempo de compilación se ejecuten las pruebas. Para la ejecución del goal de pruebas y la generación de reporte de jacoco, se puede ejecutar el siguiente comando:

mvn test jacoco:report

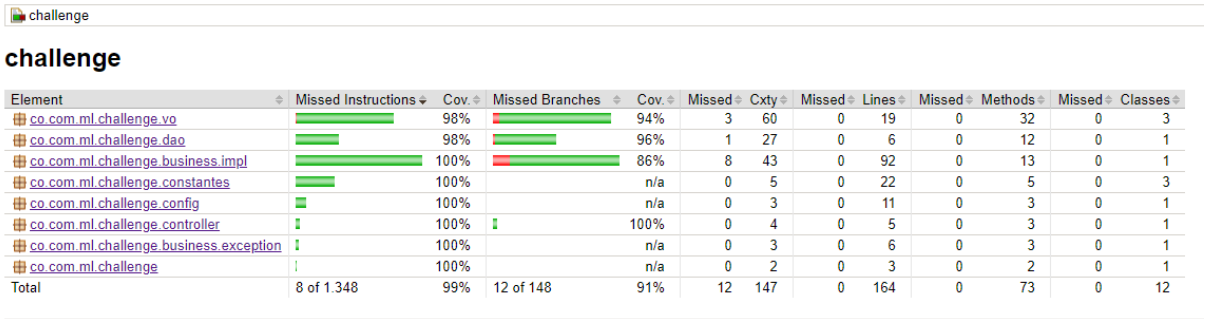
Al ejecutar este comando sobre el root del proyecto, se verifica que se ejecutan las pruebas y se genera el reporte en la ruta target del mismo:



Al finalizar la ejecución, se puede acceder al reporte en la ruta target del proyecto:



Al ingresar al reporte “index.html” se encuentra la verificación de cobertura de pruebas.

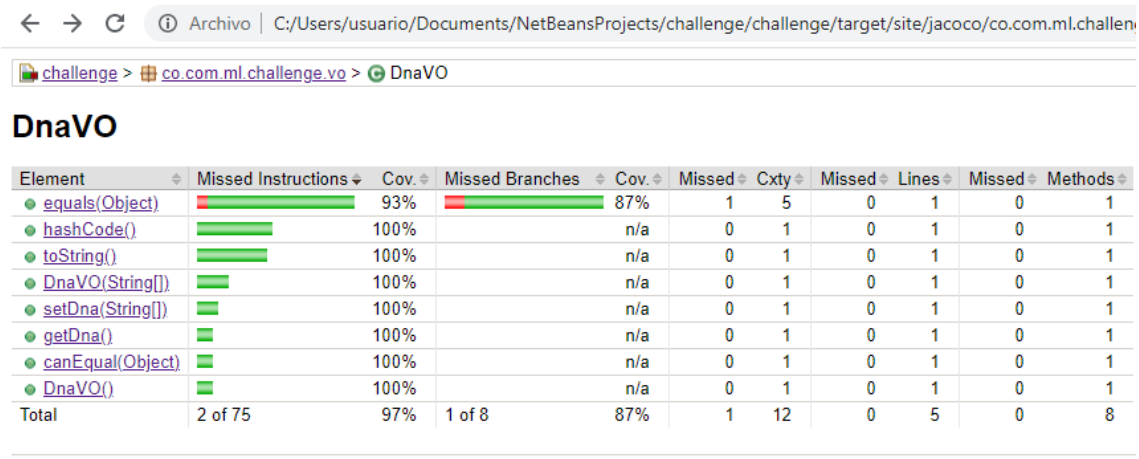


challenge

challenge

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
co.com.ml.challenge.vo		98%		94%	3	60	0	19	0	32	0	3
co.com.ml.challenge.dao		98%		96%	1	27	0	6	0	12	0	1
co.com.ml.challenge.business.impl		100%		86%	8	43	0	92	0	13	0	1
co.com.ml.challenge.constantes		100%		n/a	0	5	0	22	0	5	0	3
co.com.ml.challenge.config		100%		n/a	0	3	0	11	0	3	0	1
co.com.ml.challenge.controller		100%		100%	0	4	0	5	0	3	0	1
co.com.ml.challenge.business.exception		100%		n/a	0	3	0	6	0	3	0	1
co.com.ml.challenge		100%		n/a	0	2	0	3	0	2	0	1
Total	8 of 1.348	99%	12 of 148	91%	12	147	0	164	0	73	0	12

Para este caso identificamos una cobertura de **99% sobre las pruebas** y **91% de los branches**.



Archivo | C:/Users/usuario/Documents/NetBeansProjects/challenge/challenge/target/site/jacoco/co.com.ml.challen

challenge > co.com.ml.challenge.vo > DnaVO

DnaVO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
equals(Object)		93%		87%	1	5	0	1	0	1
hashCode()		100%		n/a	0	1	0	1	0	1
toString()		100%		n/a	0	1	0	1	0	1
DnaVO(String[])		100%		n/a	0	1	0	1	0	1
setDna(String[])		100%		n/a	0	1	0	1	0	1
getDna()		100%		n/a	0	1	0	1	0	1
canEqual(Object)		100%		n/a	0	1	0	1	0	1
DnaVO()		100%		n/a	0	1	0	1	0	1
Total	2 of 75	97%	1 of 8	87%	1	12	0	5	0	8

Aquí podemos validar que una parte del patrón builder para el método equals de los pojos no fue evaluado en las pruebas. La justificación se encuentra en el siguiente link:

https://medium.com/@mladen.bolic/lombok-data-improve-your-code-coverage-a74fb624a72b

En mi caso, preferí dejar la cobertura para identificar que si se implementaron y evaluaron las pruebas en los POJOs. No tiene un impacto funcional.

Para el caso de los branches:

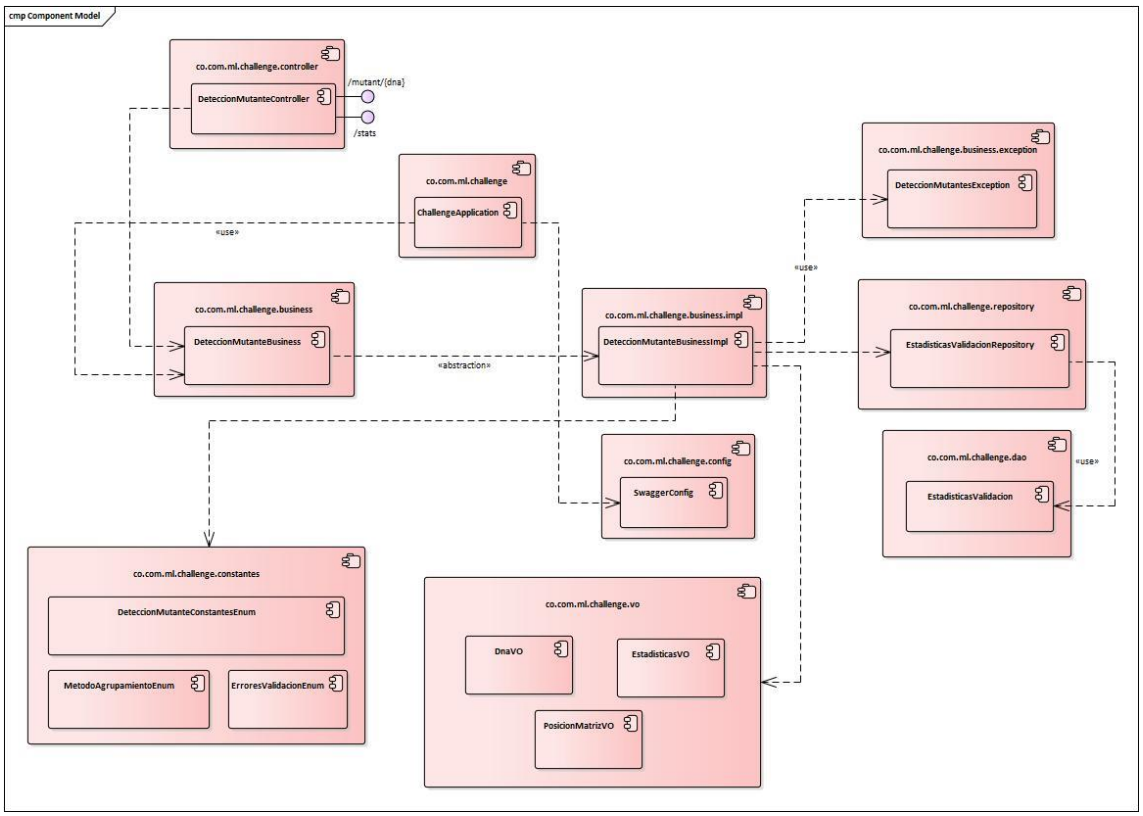
Archivo | C:/Users/usuario/Documents/NetBeansProjects/challenge/challenge/target/site/jacoco/co.com.ml.challenge.business.impl/DeteccionMutanteBusinessImpl.java.html#L136

```
49.         // Validacion de datos antes de verificar si el ADN corresponde a un mutante
50.         validaDatosADN(datos);
51.
52.         // Transformo el dato en un objeto de tipo PosicionMatrizVO y lo agrego a una coleccion de tipo LinkedList
53.         List<PosicionMatrizVO> matrizADN = new LinkedList<>();
54.         tamanioMatriz = (datos.length - 1);
55.         for (int fila = 0; fila <= tamanioMatriz; fila++) {
56.             List<String> datosFila = Arrays.asList(datos[fila].trim().split(""));
57.             int columna = 0;
58.             for (String dato : datosFila) {
59.                 PosicionMatrizVO posicionMatrizVO = new PosicionMatrizVO();
60.                 posicionMatrizVO.setFila(fila);
61.                 posicionMatrizVO.setColumna(columna);
62.                 posicionMatrizVO.setValor(dato.toUpperCase());
63.                 matrizADN.add(posicionMatrizVO);
64.                 columna++;
65.             }
66.         }
67.         // Se inicia la validacion con las reglas indicadas:
68.         boolean esMutante = false;
69.         // Iteramos por los objetos posicionados
70.         for (int index = 0; index <= tamanioMatriz; index++) {
71.             esMutante
72.                     = validacionMutante(matrizADN, index, 0, MetodoAgrupamientoEnum.HORIZONTAL)
73.                     || validacionMutante(matrizADN, 0, index, MetodoAgrupamientoEnum.VERTICAL)
74.                     || validacionMutante(matrizADN, index, 0, MetodoAgrupamientoEnum.DIAGONAL_PRINCIPAL_INFERIOR)
75.                     || validacionMutante(matrizADN, 0, index, MetodoAgrupamientoEnum.DIAGONAL_PRINCIPAL_SUPERIOR)
76.                     || validacionMutante(matrizADN, 0, (tamanioMatriz - index), MetodoAgrupamientoEnum.DIAGONAL_SECUNDARIA_SUPERIOR)
77.                     || validacionMutante(matrizADN, index, tamanioMatriz, MetodoAgrupamientoEnum.DIAGONAL_SECUNDARIA_INFERIOR);
78.             // Si cumple una condicion de mutante, no se sigue validando
79.             if (esMutante) {
80.                 break;
81.             }
82.         }
83.         log.info("Resultado de validacion: ¿Es mutante?: " + esMutante);
84.         insertaRegistroEstadistico(Arrays.toString(datos), esMutante);
85.         return esMutante;
86.     }
87.
88.     /**
89.      * Metodo principal que permite validar una matriz de datos para evaluar si
90.      * es correcta para su procesamiento Se realizan validaciones de estructura
91.      * de datos de la matriz, si tiene el tamaño minimo de comparacion, si es
92.      * cuadrada y si todos sus caracteres son validos
93.      *
94.      * @param datos
95.      */
96.     private void validaDatosADN(String[] datos) {
97.         // Calculo el tamaño del parametro, si no es el adecuado se retorna error por datos insuficientes:
98.         if (datos.length < Integer.parseInt(DeteccionMutanteConstantesEnum.TAMANIO_MINIMO_COMPARACION.valor)) {
99.             log.error(ErroresValidacionEnum.MENSAJE_TAMANIO_MATRIZ_INVALIDO.mensaje);
100.            throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, ErroresValidacionEnum.MENSAJE_TAMANIO_MATRIZ_INVALIDO.mensaje);
101.        }
102.        List<String> datosList = Arrays.asList(datos);
103.        datosList.stream().map((dato) -> Arrays.asList(dato.split(""))).forEachOrdered((letras) -> {
104.            // Verifico si la matriz es cuadrada, si no es cuadrada se retorna error por estructura invalida
105.            if (Integer.compare(letras.size(), datos.length) != 0) {
106.                log.error(String.format(ErroresValidacionEnum.MENSAJE_NO_MATRIZ_CUADRADA.mensaje, datos.length, letras.size()));
107.                throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, String.format(ErroresValidacionEnum.MENSAJE_NO_MATRIZ_CUADRADA.mensaje, datos.length, letras.size()));
108.            }
109.            // Valido si los caracteres incluidos en cada posicion de la matriz estan dentro del rango de letras valido
110.            letras.forEach((letra) -> {
111.                if (!Arrays.asList(DeteccionMutanteConstantesEnum.CARACTERES_MATRIZ_VALIDOS.valor.split(DeteccionMutanteConstantesEnum.SEPARADOR_VALORES.valor)).contains(letra.toUpperCase())) {
112.                    log.error(String.format(ErroresValidacionEnum.MENSAJE_CARACTER_INVALIDO.mensaje, letra.toUpperCase()));
113.                    throw new DeteccionMutantesException(HttpStatus.INTERNAL_SERVER_ERROR, String.format(ErroresValidacionEnum.MENSAJE_CARACTER_INVALIDO.mensaje, letra.toUpperCase()));
```

Lo que se encuentra en amarillo corresponde a los branches no validados en las pruebas, esto se resuelve a partir de la extensión de los test case con datos que evalúen todos los posibles subconjuntos descritos en el **nivel 1** del presente documento.

Con esta validación se termina el **Nivel 3** y el Challenge de Mercado Libre.

Adicional, adjunto diagrama de componentes que representa la arquitectura de solución propuesta y generada del sistema solicitado.



¡¡Muchas Gracias!!