

PRESENTACIÓN SEGUNDO LABORATORIO BASE DE DATO NoSQL 2021

Tecnólogo de Informática.

Turno Nocturno

Eduardo de los Santos

Mathías Fernandez

María Noel Costa

Índice

Elección de la base de datos	3
Tecnologías utilizadas para la solución	4
Manejo en la base de datos	4
Pruebas de los requerimientos mínimos	5
Creación de usuarios: /crearUsuario.....	5
Ingresos de roles	6
Eliminación de roles	8
Pruebas automatizadas con Jenkins	9
Prueba de carga contra RethinkDB	11
Conclusión	13
Proyecto público	13

Elección de la base de datos

Al momento de la elección de una base de datos para poder trabajar, evaluamos varios puntos, como fueran tiempo de instalación, si podía ser utilizada en diferentes sistemas operativos, si el utilizarla fuera fácil, un detalle no menor que fuera gratuito. Por todo ello, entendimos que la base de datos NoSql RethinkDB se adecuaba a todo lo que estábamos buscando.

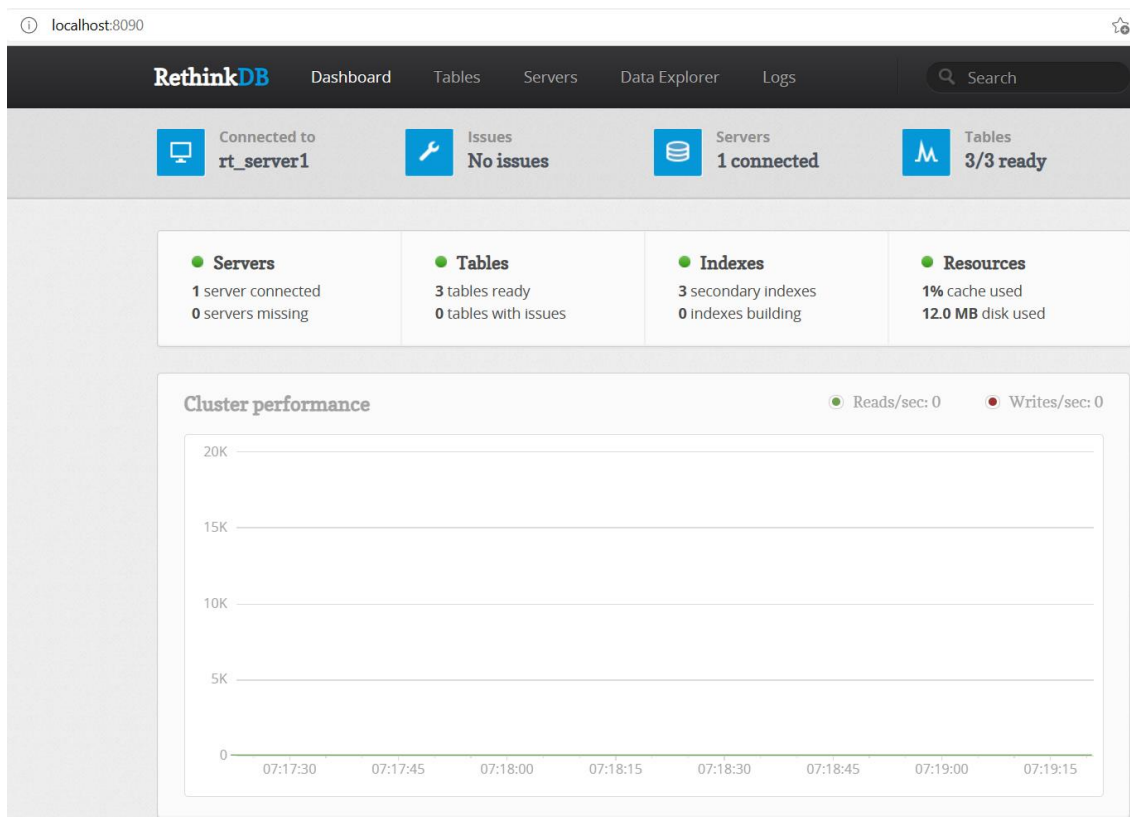
Decidimos usar Rethink dado que fue la que más nos llamó la atención de las tecnologías que mostraron por su facilidad e integridad con los lenguajes que usamos habitualmente, además esta base de datos nos facilita el transporte de datos en json.

Su uso en Windows y Springboot es muy intuitivo y la documentación es accesible para entender y manipular la base de datos con java.

Debido a que hicimos pruebas de carga la base de datos tuvo un rendimiento acorde a lo esperado, teniendo en cuenta que se está trabajando localmente se obtuvieron resultados muy buenos.

Para la instalación solo se debió descargar un ejecutable y realizar algunas modificaciones en determinados archivos.

Para poder visualizar la base de datos ingresamos en:

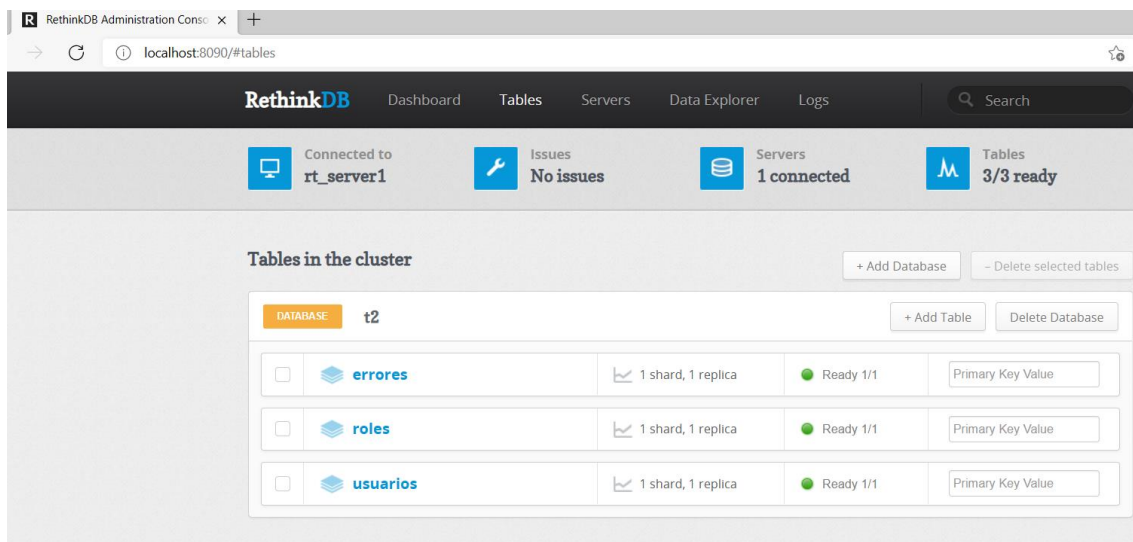


Tecnologías utilizadas para la solución

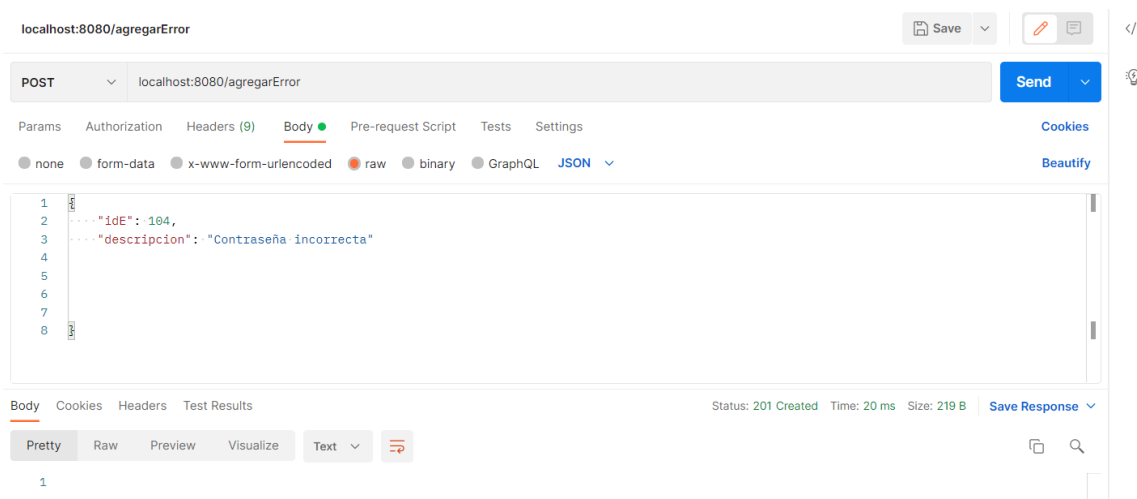
- Eclipse IDE 2021
- Springboot
- JDK11
- Rethink database 2.3
- Entorno de desarrollo Windows 10.

Manejo en la base de datos

Nuestra base de datos fue llamada como “t2”, en la misma fueron creadas las tablas necesarias para poder realizar las consultas necesarias:



Con nuestra API agregamos los errores a la base de datos de la siguiente manera, hacemos un post request: /agregarError



Entrando a la tabla errores podemos ver como quedan ingresados los datos:

Table Viewer		
Lookup: <input type="text" value="key"/>		
▲id	descripcion	idE
1327f3d5-f86...	El usuario no tiene ese rol en el sistema	103
5d86d1fe-ac2...	Contraseña incorrecta	104
71a936b3-905...	El usuario ya existe en el sistema	101
95e721c0-aa5...	El usuario no existe en el sistema	102
End of results		

Pruebas de los requerimientos mínimos

A continuación se podrá observar los pasos que se dieron para el ingreso de todas las pruebas realizadas. Se instaló el cliente de Postman para la realización de dichas pruebas, la misma es sencilla pero debes contar con una cuenta google para utilizar los servicios gratuitos que la aplicación cuenta.

Creación de usuarios: /crearUsuario

The screenshot shows a Postman interface for a POST request to `localhost:8080/crearUsuario`. The request body is a JSON object with the following fields: `correo`, `password`, `nombre`, `apellido`, and `roles`. The response is a 201 Created status with the same JSON object in the body.

```
1 {
2   "correo": "maria@gmail.com",
3   "password": "noviembre",
4   "nombre": "Maria",
5   "apellido": "Costa",
6 }
7 }
```

Body: `201 Created` Time: 1347 ms Size: 268 B

```
1 {
2   "correo": "maria@gmail.com",
3   "password": "noviembre",
4   "nombre": "Maria",
5   "apellido": "Costa",
6   "roles": []
7 }
```

La salida correcta se puede observar en la parte inferior de la imagen, que muestra los datos ingresados.

En el segundo intento de ingresar el mismo usuario, se genera el error 101:

The screenshot shows a REST client interface for a POST request to `localhost:8080/crearUsuario`. The request body is a JSON object: `{ "correo": "maria@gmail.com", "password": "noviembre", "nombre": "Maria", "apellido": "Costa" }`. The response status is `400 Bad Request` with a message: `{ "idE": 101, "descripcion": "El usuario ya existe en el sistema" }`.

```
POST localhost:8080/crearUsuario

{
  "correo": "maria@gmail.com",
  "password": "noviembre",
  "nombre": "Maria",
  "apellido": "Costa"
}
```

Status: 400 Bad Request Time: 509 ms Size: 206 B

```
{
  "idE": 101,
  "descripcion": "El usuario ya existe en el sistema"
}
```

Ingresos de roles

A los usuarios podemos ingresar de 2 maneras los roles:

De forma individual:

The screenshot shows a REST client interface for a POST request to `localhost:8080/agregarRol`. The request body is a JSON object: `{ "correoRol": "maria@gmail.com", "passwordRol": "noviembre", "nombreRol": "administrador", "roles": null }`. The response status is `200 OK`.

```
POST localhost:8080/agregarRol

{
  "correoRol": "maria@gmail.com",
  "passwordRol": "noviembre",
  "nombreRol": "administrador",
  "roles": null
}
```

Status: 200 OK Time: 96 ms Size: 262 B

En este caso el usuario va a contar con más de un rol a la vez:

The screenshot shows a REST client interface for a POST request to `localhost:8080/agregarRol`. The request body is a JSON object with the following structure:

```
1 {
2   "correoRol": "maria@gmail.com",
3   "passwordRol": "noviembre",
4   "roles": ["usuario", "supervisor"]
5 }
6
7 }
```

The response status is 200 OK, with a time of 34 ms and a size of 271 B. The response body is displayed in JSON format:

```
1 {
2   "correoRol": "maria@gmail.com",
3   "passwordRol": "noviembre",
4   "nombreRol": null,
5   "roles": [
6     "usuario",
7     "supervisor"
8   ]
9 }
```

Resultado de los roles en el mismo usuario:

The screenshot shows a REST client interface for a GET request to `localhost:8080/obtenerUsuarios`. The response status is 200 OK, with a time of 47 ms and a size of 645 B. The response body is displayed in JSON format:

```
21 {
22   "correo": "maria@gmail.com",
23   "password": "noviembre",
24   "nombre": "Maria",
25   "apellido": "Costa",
26   "roles": [
27     "usuario",
28     "supervisor",
29     "administrador"
30   ]
31 }
```

Eliminación de roles

Cuando eliminamos los roles:

The screenshot shows a REST client interface for a POST request to `localhost:8080/eliminarRol`. The request body is a JSON object with the following structure:

```
1 {
2   "correoRol": "maria@gmail.com",
3   "passwordRol": "noviembre",
4   "roles": ["usuario", "administrador"]
5 }
6
7 }
```

The response status is 200 OK, with a time of 318 ms and a size of 205 B. The response body, displayed in the 'Body' tab, is:

```
1 Roles [usuario, administrador] eliminados
```

Buscamos al mismo usuario para saber que rol le ha quedado:

The screenshot shows a REST client interface for a GET request to `localhost:8080/obtenerUsuarios`. The response status is 200 OK, with a time of 36 ms and a size of 562 B. The response body, displayed in the 'Body' tab, is a JSON object representing a user:

```
27 {
28   "correo": "maria@gmail.com",
29   "password": "noviembre",
30   "nombre": "Maria",
31   "apellido": "Costa",
32   "roles": [
33     "supervisor"
34   ]
35 }
36 }
```


Pruebas automatizadas con Jenkins

Para poder realizar pruebas automáticas con el software de Jenkins debemos haber instalado previamente JDK8 o superior y Postman.

Desde Postman se exporta una colección con las pruebas que se quieren realizar, generando un archivo .json que será utilizado por Jenkins.

La instalación del software Jenkins en Windows es sencillo; se debe instalar el plugin NodeJS.hpi que nos habilitará un entorno de ejecución para la automatización de las pruebas con la colección json.

Luego se configura Jenkins para trabajar con NodeJs y ejecutamos un comando de Windows indicando la ruta del archivo de colección json:

newman run C:\jenkins\Obligatorio2021.postman_collection.json --insecure

The screenshot shows the Jenkins configuration page for a build environment. The 'Build Environment' tab is selected, showing options for workspace management, configuration files, and NodeJS installation. The 'Build' tab is also visible, showing the command to run Newman. The 'Post-build Actions' tab is at the bottom.

Build Environment

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s)
- ☐ Provide Configuration files
- ☐ Abort the build if it's stuck
- ☐ Add timestamps to the Console Output
- ☐ Inspect build log for published Gradle build scans
- ☒ Provide Node & npm bin/ folder to PATH

NodeJS Installation:

Specify needed nodejs installation where npm installed packages will be provided to the PATH

npmrc file:

Cache location:

☐ With Ant

Build

Execute Windows batch command

Command: `newman run C:\jenkins\Obligatorio2021.postman_collection.json --insecure`

See the list of available environment variables

Advanced...

Add build step

Post-build Actions

Add post-build action

Save Apply

El resultado de la ejecución se muestra en la siguiente imagen, logrando así confirmar el correcto funcionamiento de los requerimientos mínimos solicitados.

```
Started by user mathias fernandez
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\jenkins\workspace\nosql
The recommended git tool is: NONE
No credentials specified
> git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\jenkins\workspace\nosql\.git # timeout=10
Fetching changes from the remote Git repository
> git.exe config remote.origin.url https://github.com/matf8/nsql # timeout=10
Fetching upstream changes from https://github.com/matf8/nsql
> git.exe --version # timeout=10
> git --version # 'git version 2.32.0.windows.2'
> git.exe fetch --tags --force --progress -- https://github.com/matf8/nsql +refs/heads/*:refs/remotes/origin/* # timeout=10
> git.exe rev-parse "refs/remotes/origin/main^{commit}" # timeout=10
Checking out Revision a13aa3718b9938c8cf5f6e037a7f6b0bc49cfa43 (refs/remotes/origin/main)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f a13aa3718b9938c8cf5f6e037a7f6b0bc49cfa43 # timeout=10
Commit message: "update clases"
First time build. Skipping changelog.
[nosql] $ cmd /c call C:\Windows\TEMP\jenkins3671759017537628764.bat

C:\ProgramData\Jenkins\jenkins\workspace\nosql>newman run C:\jenkins\Obligatorio2021.postman_collection.json --insecure
newman

Ob12

+ obtenerErrores
  GET http://localhost:8080/obtenerErrores [200 OK, 412B, 77ms]

+ crearUsuario
  POST http://localhost:8080/crearUsuario [201 Created, 260B, 23ms]

+ agregarRol
  POST http://localhost:8080/agregarRol [200 OK, 241B, 20ms]

+ eliminarRol
  POST http://localhost:8080/eliminarRol [200 OK, 188B, 14ms]

+ iniciarSesion
  POST http://localhost:8080/iniciarSesion [200 OK, 194B, 9ms]

+ obtenerUsuarios
  GET http://localhost:8080/obtenerUsuarios [200 OK, 1.08kB, 30ms]
```

	executed	failed
iterations	1	0
requests	6	0
test-scripts	0	0
prerequest-scripts	0	0
assertions	0	0
total run duration: 630ms		
total data received: 1.38kB (approx)		
average response time: 28ms [min: 9ms, max: 77ms, s.d.: 22ms]		

```
Finished: SUCCESS
```

Prueba de carga contra RethinkDB

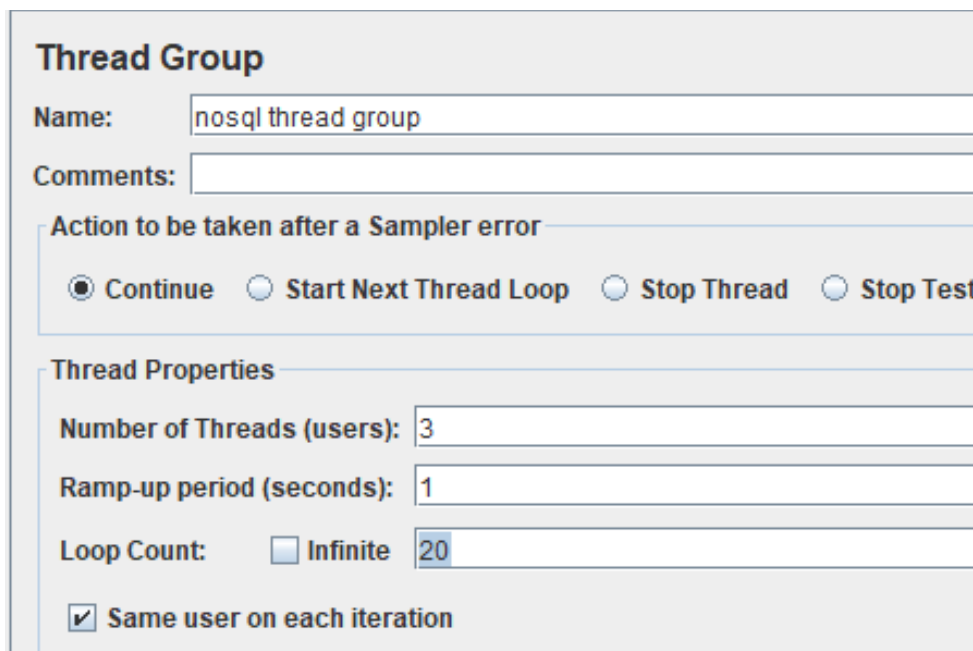
Se utilizó el software de apache JMeter 5.4.1 en Windows. En la documentación del proyecto se encuentra el archivo “Test plan - rethinkdb.jmx”, que se puede abrir con el programa y evaluar todas las pruebas.

En JMeter, se crea un Thread Group y luego se van agregando de a uno las pruebas que se consideran necesarias. En nuestro caso evaluaremos la base de datos con los servicios de obtención de errores y de usuarios, así como también servicios post de creación y adición/supresión de roles.

Para las pruebas POST, se debe configurar un HTTP Header Manager con la propiedad de Content-type = application/json.

Luego, para cada prueba se agrega un Summary Report y una tabla de resultados.

Se realizaron pruebas con una muestra según los datos de la siguiente imagen:



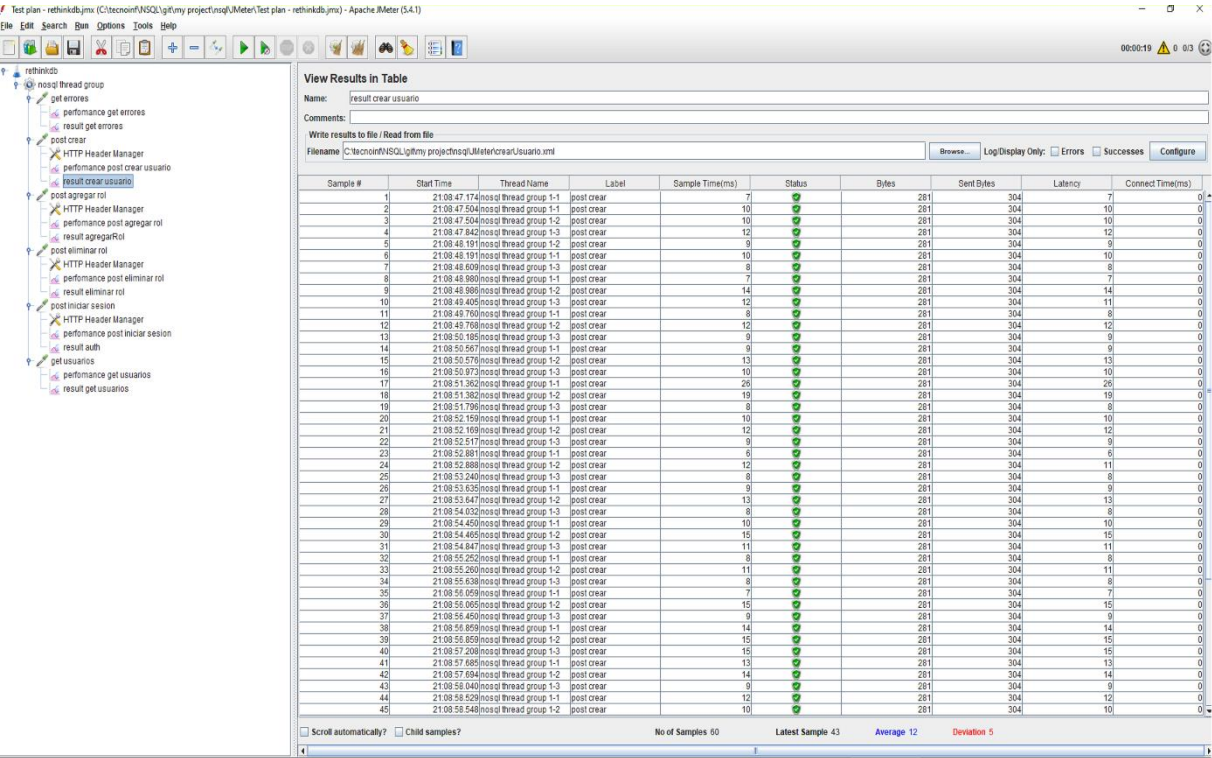
The image shows the 'Thread Group' configuration window in Apache JMeter. The 'Name' field is set to 'nosql thread group'. The 'Comments' field is empty. Under 'Action to be taken after a Sampler error', the 'Continue' radio button is selected. The 'Thread Properties' section shows 'Number of Threads (users)' set to 3, 'Ramp-up period (seconds)' set to 1, and 'Loop Count' set to 20 with the 'Infinite' checkbox unchecked. The 'Same user on each iteration' checkbox is checked.

Field	Value
Name	nosql thread group
Comments	
Action to be taken after a Sampler error	Continue
Thread Properties - Number of Threads (users)	3
Thread Properties - Ramp-up period (seconds)	1
Thread Properties - Loop Count	20
Thread Properties - Same user on each iteration	Checked

Throughput: llamada como tasa de transferencia efectiva, es el volumen de trabajo o de información neta que fluye a través de un sistema.

HTTP

Request	Path	Throughput
POST	/crearUsuario	3.8/sec
POST	/agregarRol	3.1/sec
POST	/eliminarRol	3.1/sec
GET	/getErrores	3.8/sec



Conclusión

En resumen, hemos disfrutado del desarrollo de esta solución, aprendiendo las nuevas tecnologías de NoSQL Rethink y de SpringBoot con servicios REST. El desafío planteado en el taller fue de nuestro agrado y no contamos con mayor dificultad para llegar a una solución. Con respecto a los requerimientos opcionales, nos dio la posibilidad de probar diferentes y nuevas herramientas que nos ayudara en un futuro a desenvolvernos como mejores profesionales.

Proyecto público

En el siguiente link se podrá tener acceso a los documentos y archivos necesarios para la ejecución de la solución.

[GitHub | NoSql](#)