



by



Trabajo Integrador Individual

Fecha de entrega: 12/04/2023

Mauro Raviolo


Software Developer Bootcamp - Cencosud by Alkemy

Sistema de Gestión de Reclamos

Objetivo

En el siguiente trabajo pondrás en práctica las siguientes Skills:

- Fundamentos de Nest Js
- Docker
- Terraform
- Amazon Web Service
- Bases de datos

 **Premisa:** Construir una aplicación de backend para gestionar reclamos.

Requerimientos

La aplicación debe proveer una API para realizar las siguientes operaciones:

- Agregar un nuevo reclamo con el siguiente detalle:
 - o Número de reclamo
 - o Descripción (texto)
 - o Detalle de compra (CSV formato-Fecha_compra,Nro_Factura,Cod_prod)
 - o Problema (texto)

Acciones

- Obtener una lista de todos los reclamos
- Obtener un reclamo por ID
- Actualizar un reclamo por ID
- Eliminar un reclamo por ID
- Busque reclamos haciendo coincidir una palabra clave en el título o problema.
- Obtenga una lista de reclamos filtradas por palabra clave.

La aplicación debe almacenar los reclamos en una base de datos y debe construirse utilizando NestJS, con autenticación en JWT, con un único perfil de acceso. Debe escribir pruebas para su aplicación para garantizar su funcionalidad y confiabilidad.

Bonus

- Implementar paginación de la lista de reclamos.
- Implementar la autenticación y autorización para las API para que solo los usuarios autorizados puedan agregar, actualizar y eliminar reclamos.
- Implementar una forma de cargar una imagen del producto y mostrarla junto con el resto de datos (No usar base64).

Índice

1 Consigna

3 Índice

4 Fundamentación

5 Configuración inicial

7 Estructura de las tablas

8 Instrucciones de uso

8 Inicio de sesión

10 Autenticación

11 Acciones

11 Obtener una lista de todos los reclamos

12 Obtener un reclamo por ID

13 Actualizar un reclamo por ID

14 Eliminar un reclamo por ID

15 Buscar reclamos haciendo coincidir una palabra clave en el título o problema

16 Bonus

16 Implementar paginación de la lista de reclamos

17 Implementar la autenticación y autorización para las API para que solo los usuarios autorizados puedan agregar, actualizar y eliminar reclamos

Fundamentación

El nombre elegido para la aplicación fue ***complaintsList***, dado que *complaint* en español significa “reclamo” o “queja”.

La construcción de esta aplicación de Back-End fue realizada en un entorno de **Node.js** utilizando el lenguaje **JavaScript**, a través del framework **NestJS**, el cual se basa en y sugiere el uso de **TypeScript**, un superconjunto de JavaScript que esencialmente añade tipos estáticos y objetos basados en clases.

Para la resolución de la consigna se resolvió trabajar con el lenguaje de consulta **GraphQL**, con el objetivo de interiorizar conceptos sobre este, a la vez de encontrar las formas para obtener los resultados solicitados en la letra de este trabajo integrador.

Entre las ventajas de la utilización de **GraphQL** se encuentran la posibilidad de generar APIs rápidas, flexibles y sencillas para los desarrolladores. A su vez, favorece la escalabilidad. En resumen, permite que los desarrolladores creen consultas para extraer datos de varias fuentes en una sola llamada a la API, y que creen consultas que extraigan la información necesaria para cada caso únicamente, generando un menor tiempo de respuesta y un consumo reducido de datos.

La **Base de Datos** se encuentra alojada dentro de la plataforma **Amazon Web Services** mediante **RDS**, su servicio de base de datos relacionales administrado. Esta será gestionada a través de **Postgres**, sistema que sirve para este propósito y es orientado a objetos y de código abierto. Para poder trabajar y ver la información en una interfaz amigable, se utilizó la aplicación de escritorio **TablePlus** a través de la cual es posible conectarse a la BD y visualizar su contenido.

La aplicación está pensada y creada para recibir, almacenar y devolver reclamos que pudiera generar cualquier establecimiento que se dedique a la venta de bienes y/o servicios, y sería potencialmente integrada a un **Front-End** que proveería una interfaz gráfica amigable para su uso tanto por parte de administradores del negocio, como por los clientes del mismo. Por un lado, los clientes tienen la posibilidad de registrarse y generar reclamos con toda la información necesaria para detallar el inconveniente o consulta que tengan. Por el otro, los administradores tienen acceso a la lectura de dichos reclamos. A su vez, tienen la posibilidad de modificar los reclamos (para actualizar su estado, agregar o quitar información, o eliminarlos).

Configuración inicial

A continuación se detalla una lista de pasos a través de los cuales es posible descargar la aplicación e iniciarla en entorno de desarrollo en cualquier computadora.

1. Clonar el repositorio desde:
<https://github.com/mauroraviolo23/complaints-list-cenco.git> y abrir el editor de código.
2. Copiar el archivo **.env.aws.template**, pegar la copia en el mismo directorio, renombrar a **.env** y completar las variables con los siguientes valores:

STATE=dev

DB_PASSWORD=cencotrabajo

DB_HOST=complaintslistdb.cmlzfmqgaqus.us-east-1.rds.amazonaws.com

DB_PORT=5432

DB_USERNAME=postgres

JWT_SECRET=jwt-secret-seed
3. Abrir una nueva terminal y ejecutar el siguiente comando: **yarn install**
*Esto instalará las dependencias necesarias, especificadas en el archivo **package.json***
4. Levantar la aplicación utilizando el siguiente comando: **yarn start:dev**
5. Abrir el navegador y visitar la siguiente url: **localhost:3000/graphql**
6. Una vez cargada la interfaz de **Apollo Server**, seleccionar dentro de la sección *Root Types* las siguientes instrucciones haciendo click en el símbolo de + (suma) de cada campo: **mutation** → **executeSeed**. Luego, dentro de la sección *Operation* hacer click sobre el botón azul *Mutation*. Pasados unos breves segundos, en la sección de Response debería leerse el siguiente mensaje **"Seed executed successfully"**.

Esta acción poblará la Base de Datos con “datos semilla”, información ficticia que simula ser información real. Esta herramienta es muy utilizada en instancias de desarrollo para poder trabajar y testear la aplicación sin la necesidad de estar generando datos “uno por uno”. Esta ruta debería estar protegida para que solo la puedan ver y utilizar usuarios autenticados con rol de administrador, pero para facilitar la tarea de corrección, se omitió la protección de esta.

- 7. Opcional:** Para visualizar los datos de la BD en una interfaz más amigable, abrir el programa **TablePlus**, y seleccionar la opción *Add a new connection*. Luego, seleccionar *PostgreSQL* y rellenar los campos con la siguiente información:

Name: complaintsListDB (a elección)

Host/Socket: complaintslistdb.cmlzfmqgaqus.us-east-1.rds.amazonaws.com

User: postgres

Password: cencotrabajo

A continuación hacer clic en *Connect*. Mostrará una nueva interfaz del programa, donde para ver nuestra base de datos debemos seleccionar la opción *Open a database* y elegir la llamada postgres, nombre por defecto que le da la aplicación. Ya podremos ver toda la información, mediante las dos tablas llamadas **complaints** y **users**.

Ahora que ya podemos “correr” la aplicación y visualizar la información en la BD, se explicará brevemente la estructura de cada una de las dos tablas que la conforman, y se describe cuál es el cometido de cada columna y la información que guarda.

Este paso es opcional y su único cometido es tener a total disposición toda la información presente en la Base de Datos, así como poder comprobar los cambios que se hagan al ir probando las funcionalidades. En caso de no realizar este paso, podremos ver la información que solicitemos en la respuesta de los queries que hagamos en Apollo Server.

Estructura de las tablas

Como ya fue mencionado, la Base de Datos está compuesta por dos tablas: **complaints**, donde se alojan los reclamos, y **users**, donde se alojan los usuarios que crean y/o administran tales reclamos.

La tabla **complaints** se compone de las siguientes columnas:

id: número de identificador único del reclamo

title: asunto o título del reclamo

description: explicación extendida de la causa del reclamo

details: fecha de compra, número de factura y código del producto

solved: valor booleano que indica si el reclamo fue resuelto (true) o permanece pendiente de resolución (false).

userId: número de identificador único del usuario que generó el reclamo

*La idea de generar el campo **solved** es que el administrador tenga la posibilidad de filtrar entre reclamos resueltos y pendientes, sin necesidad de tener que borrar los primeros físicamente, práctica no recomendada.*

La tabla **users** se compone de las siguientes columnas:

id: identificador único del usuario compuesto por letras y números

fullName: nombre y apellido del usuario

email: casilla de correo del usuario

password: contraseña del usuario encriptada de forma irreversible

roles: array que incluye los roles del usuario y por tanto, sus permisos. Existen dos roles posibles: *admin* y *user*.

isActive: valor booleano que indica si el usuario está activo y habilitado para utilizar la aplicación (true) o si su cuenta fue dada de baja (false)

lastUpdatedBy: identificador único del último usuario que modificó a este usuario (debe ser obligatoriamente un administrador) (puede ser nulo si no fue modificado)

*La columna de **isActive** fue creada con el fin de que se pueda "dar de baja" a un usuario de la aplicación sin necesidad de borrarlo físicamente de la BD, práctica no recomendada. Por otra parte, la columna de **lastUpdatedBy** nos da la posibilidad de saber qué usuario (administrador) modificó a este usuario por última vez, ya sea cambiando su valor de **isActive** (dicho de otra forma, activando o desactivando su cuenta) o agregándole/quitándole un rol.*

Instrucciones de uso

En las siguientes páginas se explicará cómo llevar a cabo cada una de las tareas listadas en los requerimientos de la consigna.

El primer paso para poder realizar los *query* (consultas) y *mutations* (modificaciones) es loguearnos con un usuario que tenga rol de administrador. En el seed que recientemente fue cargado existe y queda disponible un usuario con estas características, así como otros solo con las capacidades de usuario, es decir, que solo pueden crear sus propios reclamos, pero no pueden ver ni modificar los de los demás. Estas son las credenciales del usuario administrador:




email: admin@mail.com

password: 12345678

Para esto, en la interfaz de **Apollo Server** (a la que accedemos mediante la siguiente url en el navegador: localhost:3000/graphql), en la sección de *Root Types* debemos seleccionar la opción de *mutation* (haciendo clic en el signo de suma de esta), luego la opción *login*, y marcar tanto el campo de **token** como el de **user**. Esto hará que la respuesta que recibamos nos provea el token de autenticación que luego utilizaremos para validar nuestra identidad en la aplicación y así acceder a la información que se encuentra protegida.

Cuando seleccionemos el campo de **user**, nos mostrará toda la información disponible que tenemos para consultar sobre el usuario que está iniciando sesión. Por el momento, nos alcanza con elegir el *email* y el campo de *roles*, mediante los cuales sabremos que los datos que recibamos efectivamente son del usuario que queremos y que este cuenta con el rol de administrador.

Por último y antes de llevar a cabo la **mutation**, en la sección de variables debemos colocar las credenciales de inicio de sesión que fueron provistas previamente. Notar que tanto el **email** como el **password** deben ir entre comillas. Una vez hagamos clic en el botón azul de **Mutation**, en el campo de *response* obtendremos la respuesta en la que visualizaremos los datos que solicitamos, de los cuales debemos copiar el **token** para luego seguir trabajando. Así se debería ver la solicitud y la respuesta:

Response    STATUS 200 | 1.74s | 271B

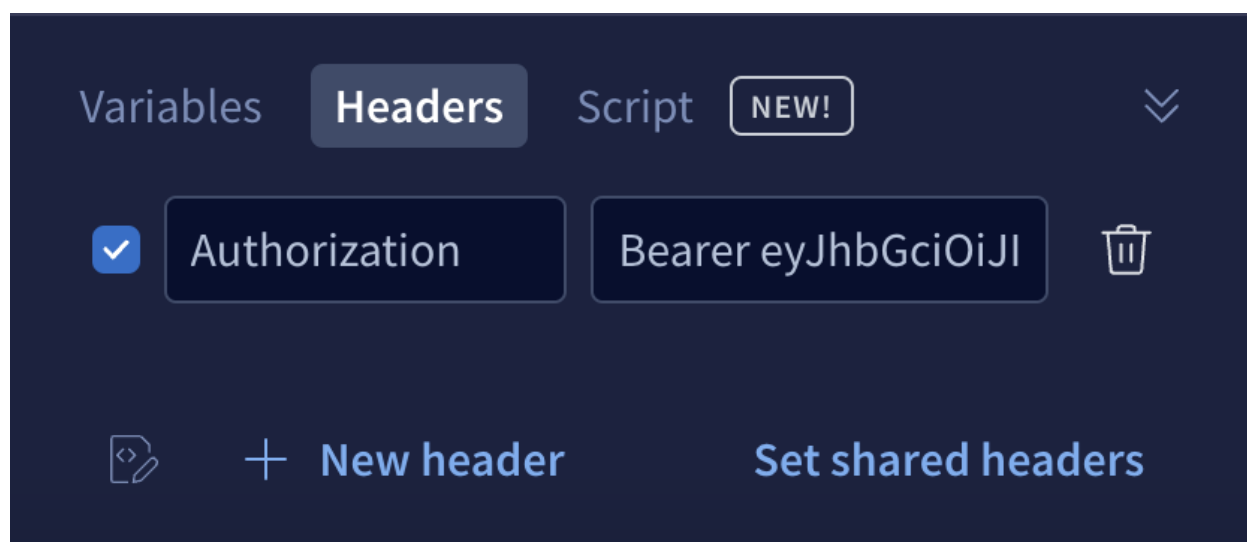
```
{
  "data": {
    "login": {
      "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpZI6ImE1OTc2YzU0LTUyOTY2NDNmNy1iNGY1LWU0ZTMxYm
NjMTRiMCI6ImIldCI6MTY4MDkxMTg2MywiZmxhIjoxNjgwOTI2
MjYzfQ.
rIobMkt6r8isSWXgw9cPivpW-VBREeu12C9I1UOuU_Q",
      "user": {
        "roles": [
          "admin"
        ],
        "email": "admin@mail.com"
      }
    }
  }
}
```

El paso que le sigue al de copiar el token desde la *response*, es guardarlo dentro de los **Headers** para que sea incluido en las próximas solicitudes que vayamos a realizar.

Las cabeceras o headers HTTP permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta. Una cabecera de petición está compuesta por su nombre (no sensible a las mayúsculas) seguido de dos puntos ':', y a continuación su valor (sin saltos de línea).

En este caso necesitamos usar el header de **Authorization**, que contiene las credenciales para autenticar a un usuario en un servidor. Para esto, necesitamos seleccionar la opción **Headers**, y a continuación **New header**. Esto nos habilitará dos campos para ingresar información. El primero, que figura como *header key*, es donde se debe seleccionar "Authorization". En el campo value es donde debe ir el token que recibimos en la respuesta anterior, precedido de la palabra "Bearer". Por último debemos asegurarnos de que este header quedó seleccionado, marcando el checkbox que se encuentra a la izquierda de los campos que acabamos de rellenar.

Así es como se vería esta tarea luego de completada:



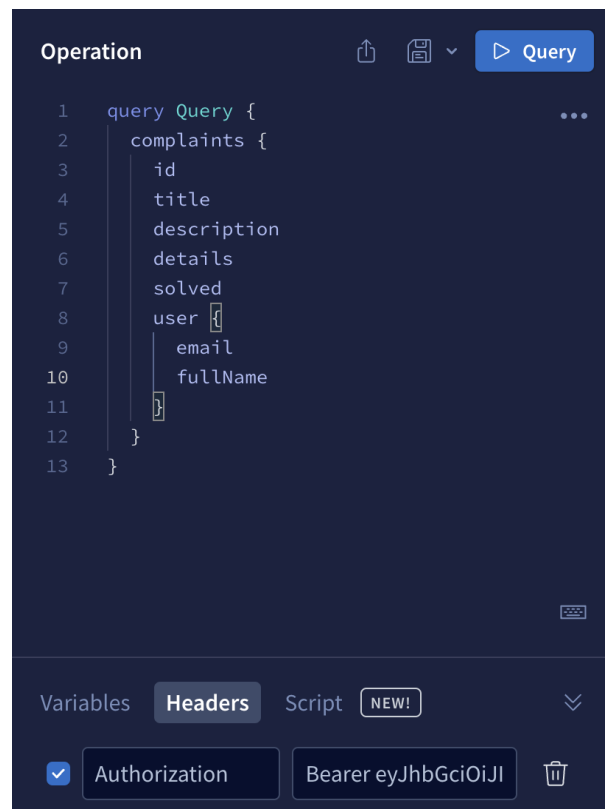
¡Listo! Ahora en cada solicitud que hagamos la aplicación nos reconocerá como usuarios autenticados.

*Es importante aclarar que esta es solo la "puerta de entrada" a la aplicación, pero la posibilidad de utilizar ciertas o funcionalidades o no, está subordinada al rol que tenga asignado cada usuario. La validación de estos roles está a cargo de otro componente del código, que utiliza el mismo token para saber qué usuario es el que está realizando la solicitud, dado que en el payload o cuerpo de este token se encuentra el **id** del usuario, mediante el cual luego podemos conocer sus datos, entre ellos los roles que tiene asignados.*

Acciones - Obtener una lista de todos los reclamos

Debemos abrir una nueva pestaña en el **Apollo Server** y seguir los siguientes pasos:

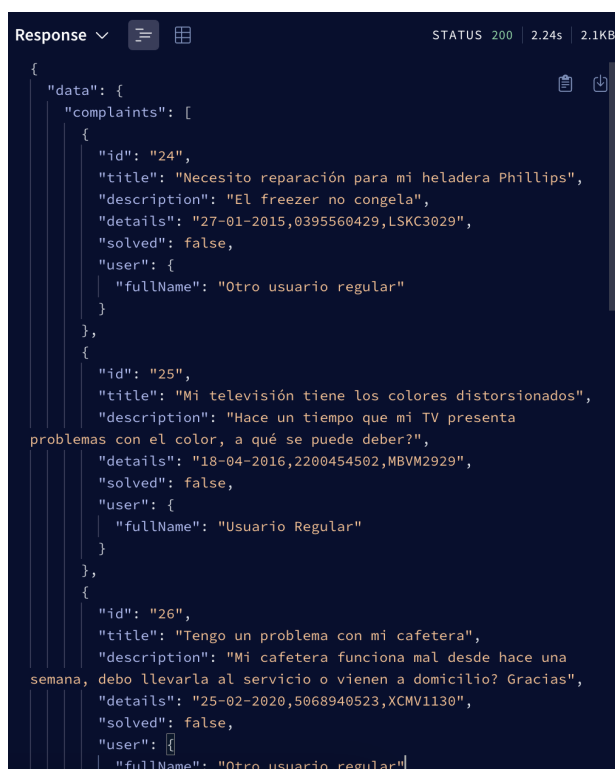
1. Seleccionar la opción *query*
2. Seleccionar la opción *complaints* (en plural)
3. Dentro del apartado **Fields**, elegir los campos de cada reclamo que quiero recibir (uno de esos campos debería ser el de *user* para saber a quién pertenece)
4. Haciendo clic en el campo *user*, se despliega la lista de los campos del usuario que se puede elegir recibir (con el *email* y *fullName* ya es suficiente para identificarlo)
5. Correr la solicitud haciendo clic en el botón azul **Query**



Al estar construyendo la query podemos ver que tenemos disponible una sección de **Arguments**: esta cuenta con tres argumentos opcionales que sirven para realizar la paginación de los reclamos o buscar por una palabra específica; podemos ignorarlos por ahora.

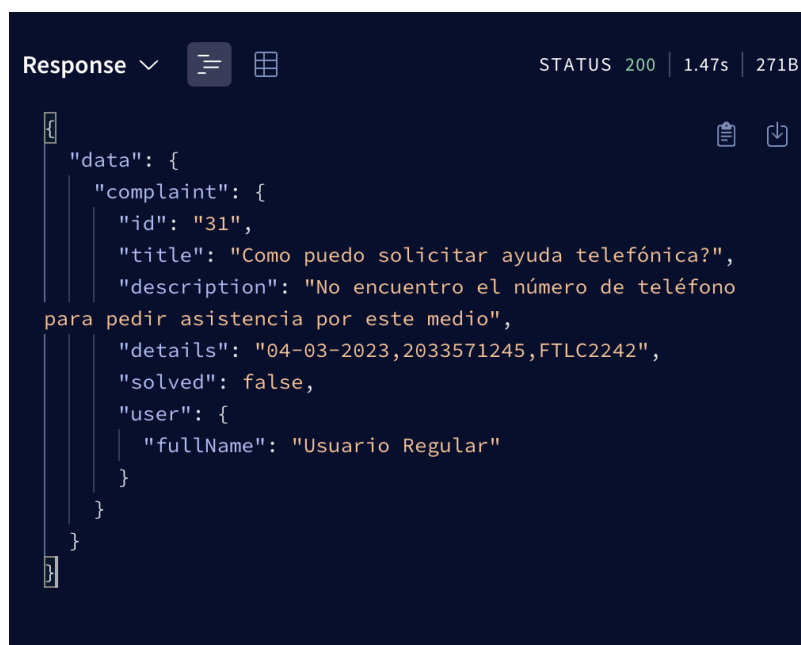
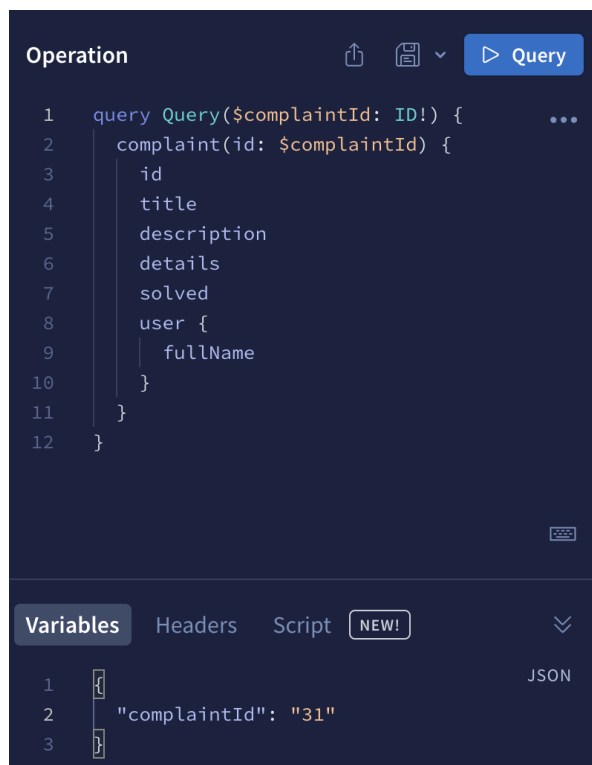
Notar que dentro de los Headers se envía el token mediante el que se verificará nuestra identidad y por tanto, si estamos autorizados a recibir la lista de reclamos. Esto también significa que un usuario sin rol de administrador **no** estará autorizado a recibir esta información.

Como podemos ver, la respuesta será un listado de todos los reclamos de todos los usuarios, en el orden en el que fueron ingresados a la Base de Datos.



Acciones - Obtener un reclamo por ID

1. Seleccionar la opción *query*
2. Seleccionar la opción *complaint* (en singular)
3. Dentro del apartado **Fields**, elegir los campos del reclamo que quiero recibir (uno de esos campos debería ser el de *user* para saber a quién pertenece)
4. Haciendo clic en el campo *user*, se despliega la lista de los campos del usuario que se puede elegir recibir (con el email y *fullName* ya es suficiente para identificarlo)
5. En la sección **Variables** veremos que figura un objeto con la llave ***complaintId*** y el valor en *null*. Cambiar ese valor por el número identificador del reclamo.
6. Correr la solicitud haciendo clic en el botón azul **Query**



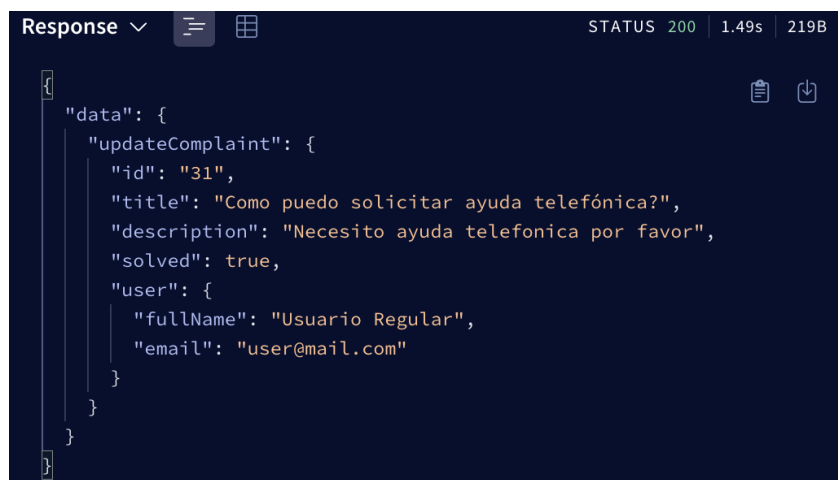
En el cuerpo de la respuesta vemos la información del reclamo.

En caso de enviar un número de reclamo que no existe, la respuesta recibida sería "Complaint with id:*número de id* not found".

De la misma forma que para la solicitud anterior, un usuario sin rol de administrador **no** podrá realizar esta solicitud.

Acciones - Actualizar un reclamo por ID

1. Seleccionar la opción *mutation*
2. Seleccionar la opción *updateComplaint*
3. Dentro del apartado **Fields**, elegir los campos del reclamo que quiero recibir en la respuesta (idealmente debería ser al menos los que voy a actualizar para visualizar el cambio)
4. Haciendo clic en el campo *user*, podemos elegir la lista de los campos del usuario que queremos recibir (el *email* y/o *fullName* ya es suficiente para conocer a quién pertenece el reclamo)
5. Volviendo hacia atrás y en la sección de **Arguments**, seleccionando *updateComplaintInput* podemos marcar los campos que queremos actualizar, que se agregarán en la sección de **Variables**, en principio con valor *null*. Debemos modificarlo por el valor real que queremos que ahora tenga. El único que obligatoriamente debemos elegir y proveer es el *id*, para identificar el reclamo que actualizaremos.
6. Correr la solicitud haciendo clic en el botón azul **Mutation**

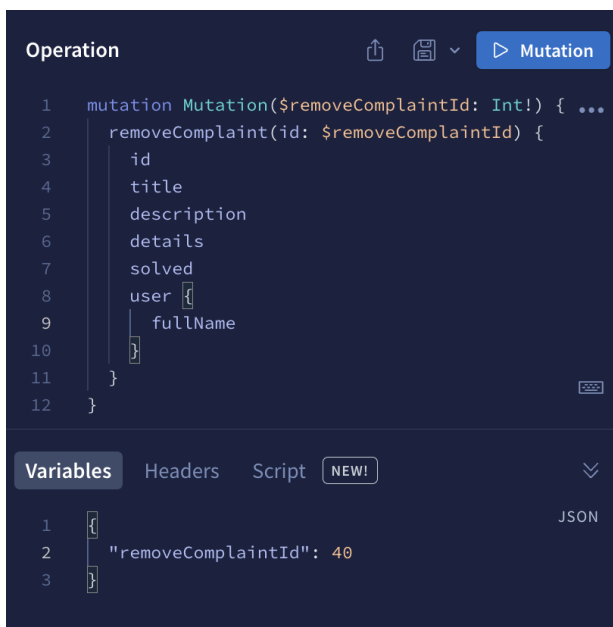


En este ejemplo, un usuario administrador decidió actualizar el campo **solved** (cambiar su valor a *true*) para establecer ese reclamo como resuelto.

Si un usuario regular provee un *id* de reclamo que no existe o no le pertenece, recibirá como respuesta "Complaint with id:*número de id* not found".

Acciones - Eliminar un reclamo por ID

1. Seleccionar la opción *mutation*
2. Seleccionar la opción *removeComplaint*
3. Dentro del apartado **Fields**, elegir los campos del reclamo que quiero recibir en la respuesta
4. En la sección **Variables**, figurará *removeComplaintInput* con valor *null*. Sustituir por el número de reclamo a borrar.
5. Correr la solicitud haciendo clic en el botón azul **Mutation**



En la sección **Response** recibiremos la respuesta que contiene los campos que seleccionamos en el paso 3.

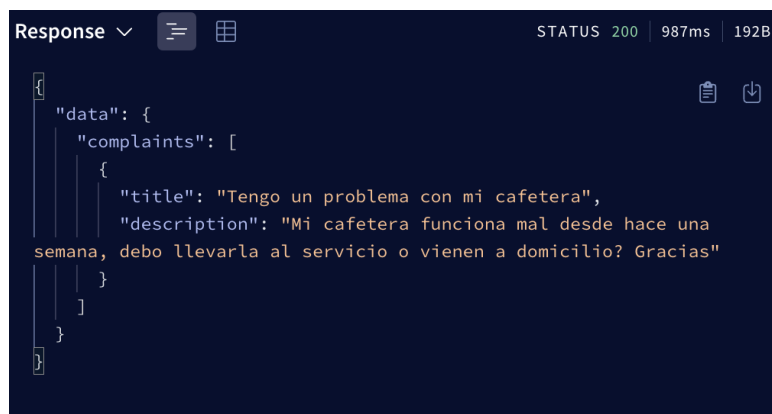
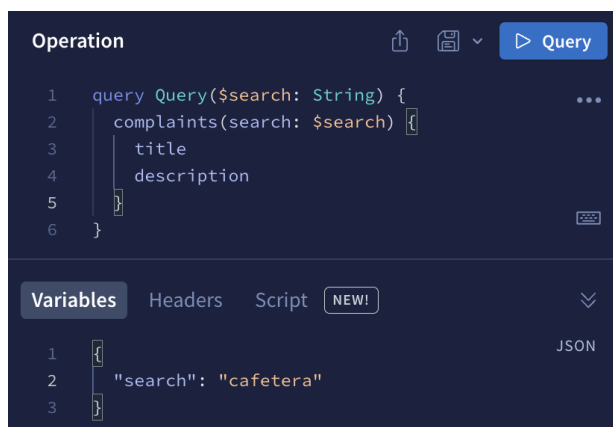
Además en la siguiente imagen podemos observar la respuesta en caso de enviar un **id** de reclamo inexistente, o en caso de que un usuario sin rol de administrador envíe un id de reclamo que no le pertenece.



Una característica que tiene esta opción de eliminar un reclamo, es que un usuario sin rol de administrador **no** elimina físicamente sus reclamos, sino que les cambia su estado a resuelto (solved pasa de false a true).

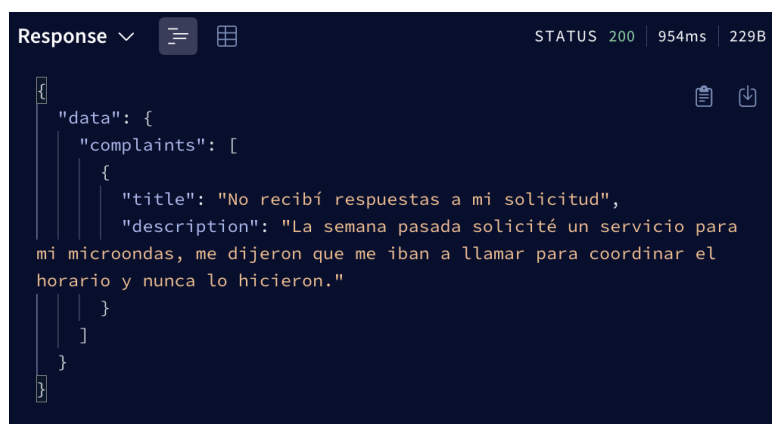
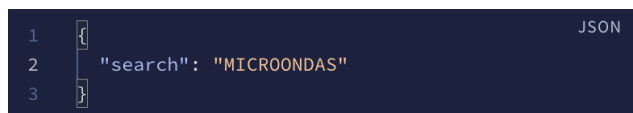
Acciones - Buscar reclamos haciendo coincidir una palabra clave en el título o problema

1. Seleccionar la opción *query*
2. Seleccionar la opción *complaints*
3. Dentro del apartado **Fields**, seleccionar los campos que quiero recibir de cada reclamo
4. Dentro del apartado **Arguments**, seleccionar la opción *search*.
5. En la parte de Variables, ahora aparecerá "search" con valor *null*. Sustituir por la palabra clave que quiero que esté presente en los reclamos que reciba
6. Correr la solicitud haciendo clic en el botón azul **Query**



La respuesta nos devuelve una lista de el o los reclamos que coinciden con la palabra clave que ingresamos.

Es importante tener en cuenta que esta palabra clave puede estar presente en el **título** y/o en la **descripción** del reclamo. Además, **no es sensible a las mayúsculas**, lo que flexibiliza la búsqueda y ayuda a obtener los resultados.



En este otro ejemplo la palabra clave buscada está presente en la descripción del reclamo, y como podemos ver no importa si fue escrita en mayúsculas o minúsculas.

Bonus - Implementar paginación de la lista de reclamos

1. Seleccionar la opción *query*
2. Seleccionar la opción *complaints*
3. Dentro del apartado **Fields**, seleccionar los campos de cada reclamo que quiero recibir
4. Dentro del apartado **Arguments**, seleccionar *offset* y/o *limit*:

offset: para indicar cuántos reclamos saltará la respuesta, o sea, si el valor es "5", omitirá los primeros cinco reclamos y listará a partir del sexto (por defecto es 0)

limit: indica el número máximo de reclamos a devolver (por defecto es 20)

5. En la sección **Variables**, sustituir los valores *null* por los valores elegidos para la consulta
6. Correr la consulta haciendo clic en el botón azul **Query**



Como podemos observar en el cuerpo de la respuesta, la lista que recibimos saltó los primeros **dos** reclamos registrados en la Base de Datos, y además nos devolvió un máximo de **tres** reclamos, tal como le solicitamos.

Si el número de reclamos que saltará es igual o mayor a la cantidad total de reclamos existentes, nos devolverá una lista vacía.

Si el número de reclamos a devolver es menor al límite solicitado, los devolverá todos.

Bonus - Implementar la autenticación y autorización para las API para que solo los usuarios autorizados puedan agregar, actualizar y eliminar reclamos

Si bien en la consigna del ejercicio este punto está demarcado como un **Bonus**, es decir, como una funcionalidad “extra” y opcional de la aplicación, se decidió que esta funcionalidad sea parte de la creación del proyecto desde el inicio.

Como primera “barrera” entre el mundo exterior y la aplicación tenemos la **autenticación** de usuarios. Es necesario estar registrado e iniciar sesión para poder utilizar cualquiera de las funcionalidades. Sin estar autenticado, no se estará autorizado a trabajar con la aplicación.

Exceptuando la ejecución del Seed (página 5 y 6), planteado así únicamente para facilitar la tarea de corrección.

¿Cómo se inicia sesión? Enviando nuestras credenciales al sitio al que queremos ingresar. Es necesario haberse registrado previamente.

*Esto está explicado en las páginas 8, 9 y 10. En caso de querer crear un usuario nuevo, en lugar de usar la ruta de `login`, debemos usar la ruta de `signUp`, añadiendo al cuerpo de nuestra solicitud nuestro `fullName`. Notar que este usuario que creemos tendrá, por defecto, rol de **user**.*

¿Cómo se verifica que un usuario haya iniciado sesión? Mediante la autenticación por JWT Token. Cuando un usuario se autentica en el sitio, el servidor genera un token JWT y lo envía al cliente, quien lo almacena en su dispositivo. Cada vez que el cliente hace una solicitud al servidor, incluye el token JWT en el encabezado de la solicitud.

El servidor, al recibir la solicitud, extrae el token JWT del encabezado y verifica si es válido. Para ello, comprueba la firma digital del token usando la clave secreta del servidor y comprueba si el token ha caducado o si ha sido revocado. Si el token es válido, el servidor procesa la solicitud y devuelve la respuesta correspondiente.

*Podemos ver el paso a paso de la recepción y almacenamiento de este token en la página 10. Además aquí mencionamos por primera vez que **el token puede “caducar”**, es decir, tiene un tiempo máximo de duración (modificable por quien crea la aplicación) luego del cual será necesario volver a iniciar sesión y generar uno nuevo.*

Como ya fue mencionado anteriormente, existen dos tipos de roles: **user** y **admin**. Esto nos introduce al apartado de la **autorización**. Dependiendo del rol que tenga el usuario, estará -o no- autorizado a realizar distintas acciones dentro de la aplicación. Cuando no se esté autorizado a realizar determinada acción se recibirá un mensaje de error en el que se especificará la razón por la que se le prohíbe proceder (normalmente será un mensaje que describe el rol que debería tener el usuario para estar autorizado).

Veamos listadas las funciones para las que está autorizado cada tipo de usuario:

admin:

- Ver el listado de todos los reclamos
- Buscar cualquier reclamo por ID
- Actualizar cualquier reclamo por ID
- Eliminar cualquier reclamo por ID

user:

- Ver el listado de todos **sus** reclamos
- Buscar cualquiera de **sus** reclamos por ID
- Actualizar cualquiera de **sus** reclamos por ID
- Eliminar cualquiera de **sus** reclamos por ID (no borrado físico)

*En resumen, un usuario con rol de **user** solo podrá visualizar y modificar reclamos que le pertenezcan, mientras que un usuario **administrador** podrá hacerlo con cualquier reclamo, independientemente de a quién pertenezca.*

Llegamos al final de este informe.

Espero que sea del agrado de quien lo lea. El enfoque principal fue intentar crear un reporte “amigable” del trabajo realizado, junto a explicaciones lo más claras posible, que no requieran de un background técnico tan amplio para poder comprender cómo funciona todo. A la vez, funcionó como una instancia para despejar dudas y reforzar los conocimientos.

Lograr la concreción de esta tarea no hubiera sido posible sin el apoyo de mis compañeros.

Muchas gracias por leer.

Mauro Raviolo

Software Developer Bootcamp - Cencosud by Alkemy