



El futuro digital
es de todos

MinTIC



Ciclo 4a:

Desarrollo de aplicaciones web



**Misión
TIC2022**

VERSIÓN 1.0

Unidad de educación
continua y permanente
Facultad de Ingeniería



Unidad Camilo Torres
Calle 44 # 45-67
Bloque 85 piso 1



(57) + 316 5000
uec_fibog@unateduco

Actividad Práctica

(Despliegue y Pruebas del API Gateway)

Antes de probar el API Gateway, este se desplegará en Heroku utilizando Docker, de la misma forma en que se desplegaron los microservicios [auth_ms](#) y [account_ms](#). Posteriormente, para asegurarse de que el despliegue y que la exposición de funcionalidades se hizo correctamente, se utilizará el cliente GraphQL dispuesto por Apollo.

Despliegue del API Gateway en Heroku

Para desplegar el API Gateway en Heroku, se deben seguir los mismos pasos utilizados para el despliegue de los microservicios [auth_ms](#) y [account_ms](#). Por ello, se debe crear un [Dockerfile](#) en la raíz del proyecto con los siguientes comandos:

```
FROM node:14
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 4000
CMD [ "node", "src/index.js" ]
```

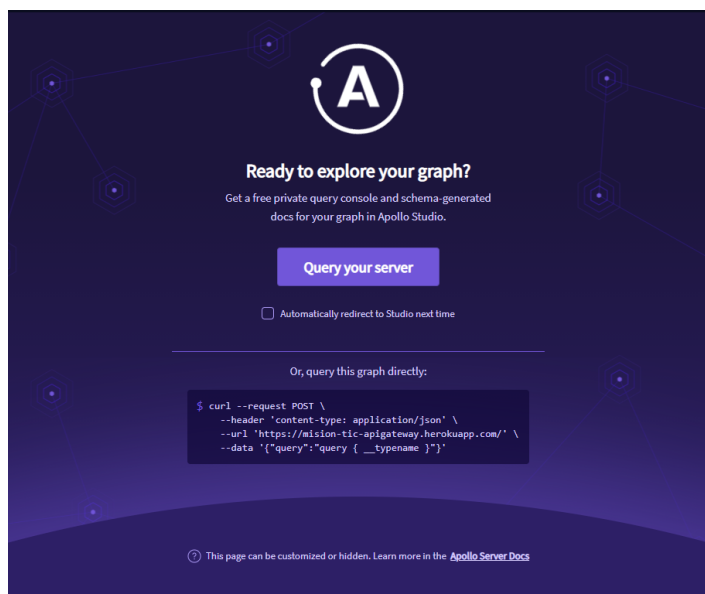
Posteriormente, si se encuentra en Windows o MacOS se debe abrir Docker Desktop, y en la terminal se deben utilizar los siguientes comandos de Heroku en orden:

```
heroku login
heroku create appName
heroku container:login
heroku container:push web --app appName
heroku container:release web --app appName
```

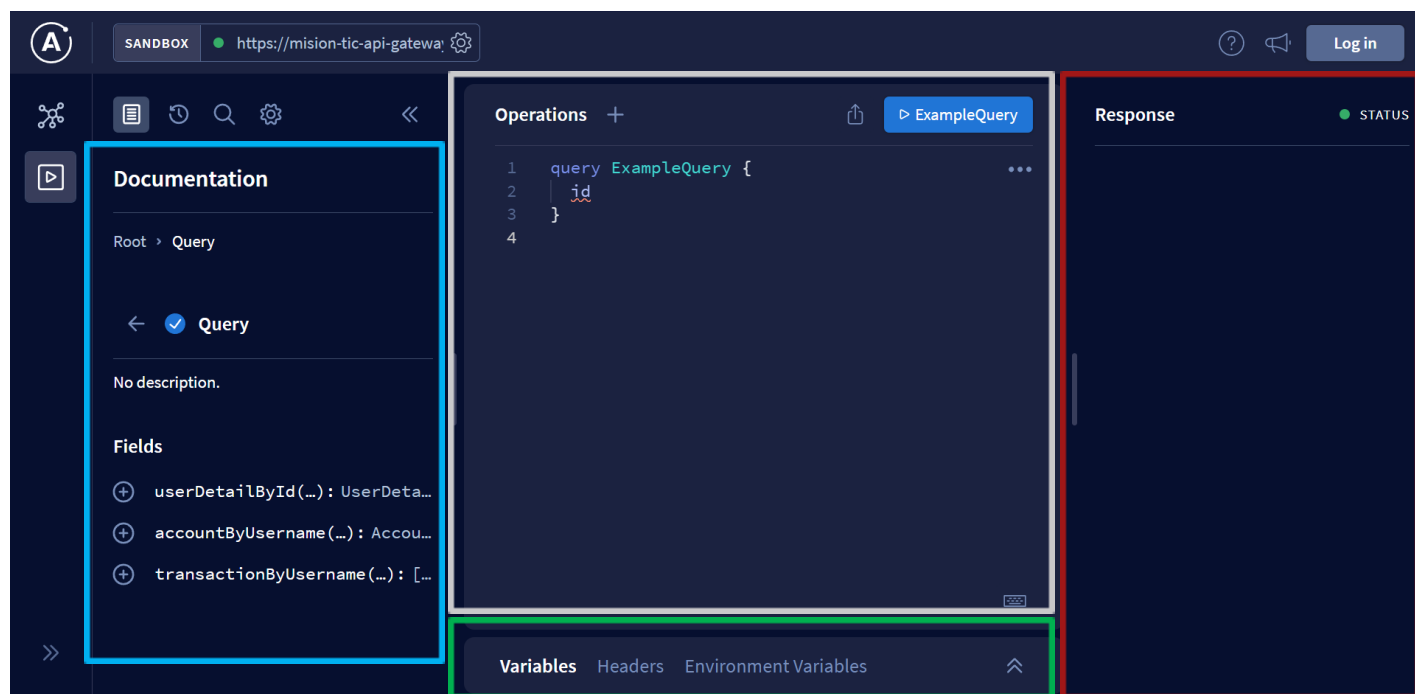
Nota: de ser necesario, en el material de la clase se encuentra un archivo llamado [C4a.AP.09.api_gateway.rar](#), que contiene el [Dockerfile](#) creado anteriormente.

Pruebas del API Gateway

Para probar las funcionalidades del API Gateway, se debe ingresar a la aplicación de Heroku con el comando `heroku open --app appName`:



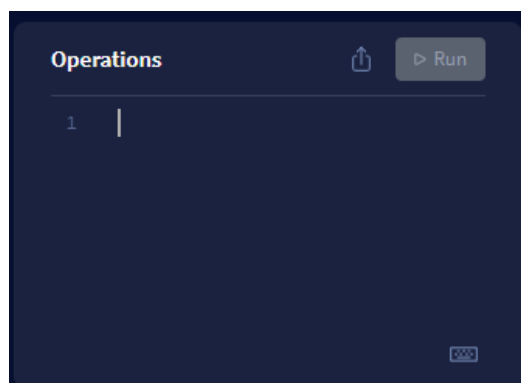
En la ventana, se presiona el botón `Query your server`. Esto redireccionará al cliente GraphQL dispuesto por Apollo, donde se realizarán las pruebas sobre las peticiones del API Gateway:



En la interfaz de la herramienta se pueden ver cuatro secciones:

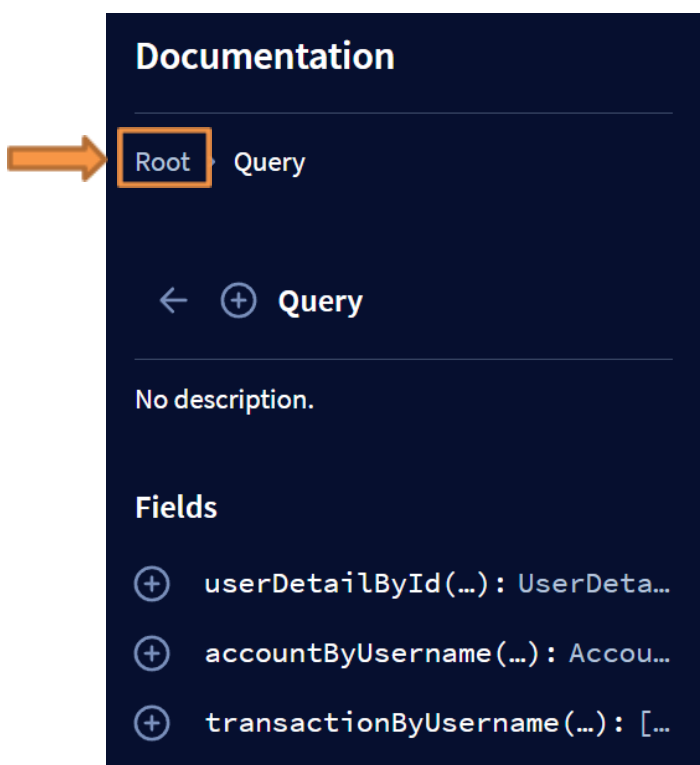
- Azul: la sección **Documentation** sirve para consultar el formato de las **Queries** y **Mutations** dispuestas por el API, y para crear las peticiones a partir de dicho formato.
- Gris: la sección **Operations** sirve para ver, editar y enviar la petición al API Gateway.
- Verde: la sección **Variables/Headers/Environment Variables** es aquel lugar en el cliente donde se definen cada uno de estos valores. Como se verá más adelante, para probar el API Gateway, únicamente se utilizarán las subsecciones **Variables** y **Headers**.
- Rojo: la sección **Response**, es el lugar donde se mostrará la respuesta del API a una petición realizada.

Inicialmente, Apollo crea una **Query** de ejemplo en la sección **Operations**. Ya que esta petición no se va a utilizar, se debe eliminar manualmente:

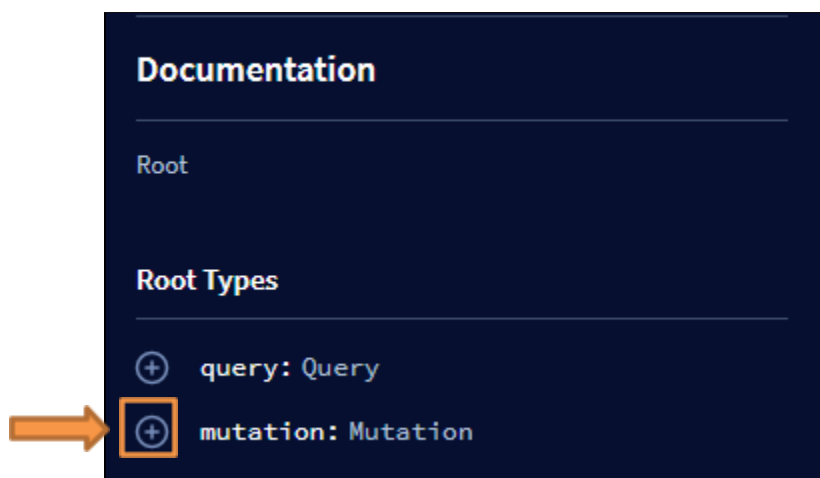


Generalmente, cuando se prueba una API GraphQL, se pueden construir las peticiones de dos formas: escribiéndolas manualmente, o utilizando alguna herramienta que lo haga automáticamente. El Sandbox de Apollo ofrece una herramienta que permite la rápida construcción de las peticiones, esta se encuentra en la sección [Documentation](#). Para construir una petición basta con seleccionar y/o deseleccionar la casilla junto a cada campo descrito en la documentación, según se requiera.

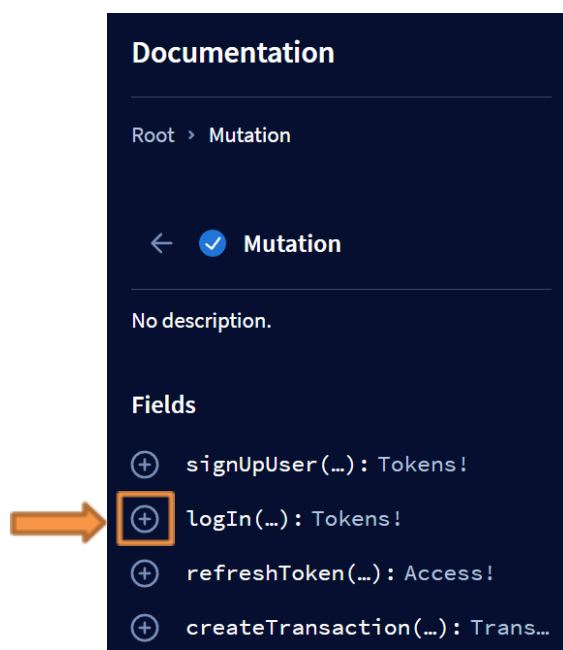
Por ejemplo, para hacer la [Mutation login](#), se debe dirigir al nivel [Root](#):



Y allí se debe seleccionar la casilla junto a [mutation](#):

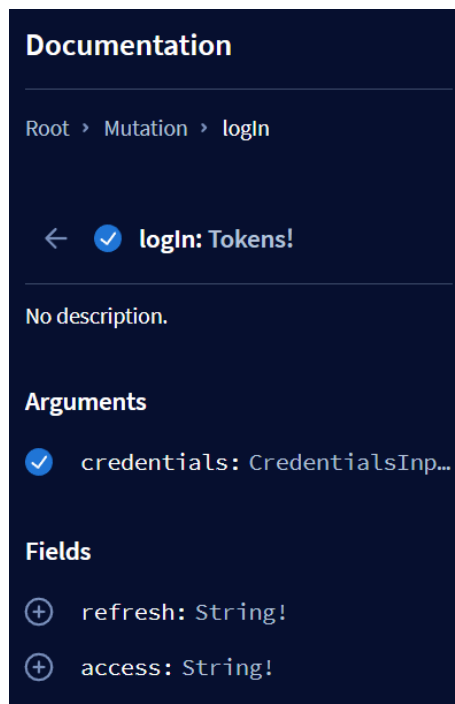


Esto redirige al nivel [Mutation](#), donde se encuentran todas las mutaciones declaradas en el API Gateway. Dado que se va a realizar una autenticación, se marca la casilla junto a [login](#):



Luego, en el nivel [login](#) se muestran tanto los parámetros de la petición, como los campos que retorna el API Gateway. Dado que el argumento [credentials](#) se marcó en el API Gateway como un argumento

obligatorio en el *TypeDef* de la *Mutation*, la herramienta lo selecciona automáticamente:



Como se ha mencionado anteriormente, una ventaja que tiene utilizar GraphQL es que permite al cliente elegir los datos que se desea recibir en la respuesta de la petición. En este caso, se desea recibir tanto el *access token* como el *refresh token*, por lo cual, se den seleccionan ambos campos (*Fields*):

Documentation

Root › Mutation › logIn

← ☒ **logIn: Tokens!**

No description.

Arguments

☒ **credentials: CredentialsInp...**

Fields

☒ **refresh: String!**

☒ **access: String!**

De esta forma, la construcción de la petición en la sección [Operations](#), se debe ver de la siguiente forma:

Operations + 🔗 ▶ Mutation

```

1  mutation Mutation($credentials: CredentialsInput!) {
2    logIn(credentials: $credentials) {
3      refresh
4      access
5    }
6  }

```

Sin embargo, aún no se puede realizar la petición, pues se deben ingresar los valores del argumento de la petición. Para ello, en la sección [Variables](#), que se encuentra debajo de la sección [Operations](#), Apollo crea una variable inicializada con `null`:

Variables Headers Environment Variables

```

1  {
2    "credentials": null
3  }

```

JSON

Para ingresar los valores del argumento de la petición, se debe reemplazar *null*, por un formato JSON que contenga las llaves indicadas en el *TypeDef* y los valores que se deseen ingresar. El formato se puede consultar ingresando a la descripción del argumento (*credentials*) en la sección *Documentation*:

Arguments

☒ *credentials: CredentialsInput!* →

Fields

☒ *refresh: String!*

☒ *access: String!*

← *credentials: CredentialsInput!*

No description.

Input Fields

username: String!

password: String!

De esta forma, un formato JSON válido con valores que permitirán realizar la autenticación es el siguiente:

Variables Headers Environment Variables

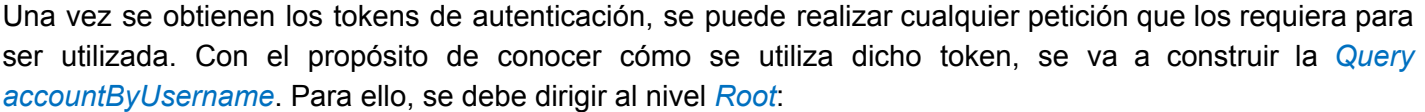
```

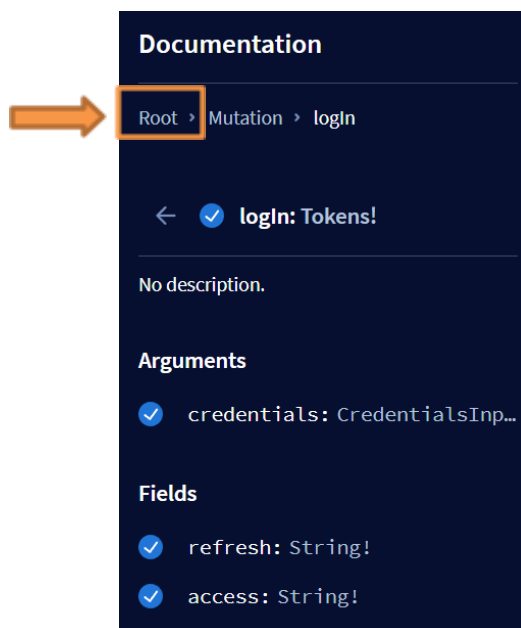
1  {
2    "credentials": {
3      "username": "gaortegagb",
4      "password": "contraseña"
5    }
6  }

```

JSON

Finalmente, se realiza la petición pulsando el botón *Mutation*, que se encuentra en la parte superior derecha de la sección *Operations*. La respuesta que se obtiene es la siguiente





Se debe deseleccionar la casilla asociada con *mutation*, y se debe eliminar la variable *credentials*. Luego, se deben seleccionar en orden los componentes necesarios para la construcción de la *Query* *accountByUsername*. Al hacerlo, la sentencia de la petición en la sección *Operations* debe ser la siguiente:

```

Operations +
1 query Query($username: String!) {
2   accountByUsername(username: $username) {
3     username
4     balance
5     lastChange
6   }
7 }

```

Y el contenido de las variables debe ser similar a lo siguiente:

Variables Headers Environment Variables


```

1  {
2  "username": "gaortegagb"
3  }
    
```

JSON


Ahora, para añadir el token a la petición, se debe dirigir a la pestaña [Headers](#), que se encuentra junto a [Variables](#). Allí se debe pulsar el botón [New header](#):

Variables **Headers** Environment Variables

 [+ New header](#) [Set default headers](#)

En el campo que se muestra, en la [header key](#) se debe seleccionar la opción [Authorization](#), y en [value](#) se debe ingresar el [access token](#) obtenido anteriormente (Sin comillas):

Variables **Headers** Environment Variables

☒ [Authorization](#) 

[+ New header](#) [Set default headers](#)

Con esto, ya se puede enviar la petición al API Gateway. Al hacerlo, se obtiene la respuesta adecuada:

Response ▾ [Menu Icon] [Grid Icon] ● STATUS 200 | 27.8s | 119B

```
{
  "data": {
    "accountByUsername": {
      "username": "gaortegagb",
      "balance": 990000,
      "lastChange": "2021-11-02T20:57:58.459+00:00"
    }
  }
}
```

Si se recibe un valor *null*, se debe a que el *username* ingresado no coincide con el *username* del usuario al que le pertenece el token ingresado en los *headers*. Utilizando estos conceptos, es posible probar las demás funcionalidades ofrecidas por el API Gateway.