



**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 Camillo Torres
Calle 44 # 45-67
Bloque B5 piso 1



(57) + 316 5000
vec_itbog@unatedu.co

Actividad Práctica

(Despliegue de Microservicio AccountMS)

En sesiones anteriores, se desarrolló el microservicio [account_ms](#) utilizando [Spring Boot](#). En esta guía se realizará el despliegue de este microservicio en Heroku, utilizando Docker.

Configurar el microservicio para el despliegue

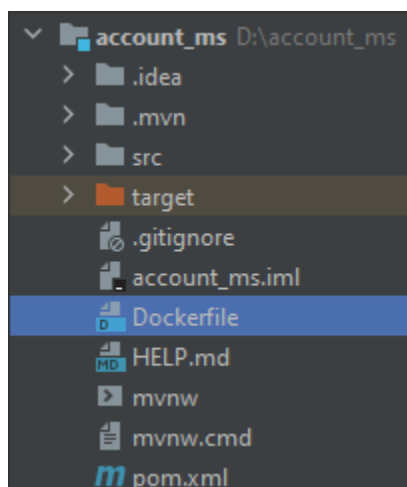
Para desplegar el microservicio, es necesario realizar algunas configuraciones primero, una de ellas es añadir el código `server.port=${PORT:8080}` al archivo `src/main/resources/application.properties`. Al hacerlo, el archivo debe tener un código similar al siguiente:

```
spring.data.mongodb.uri=${URL_DB: url}
server.port=${PORT:8080}
```

Luego, es necesario empaquetar el proyecto en un `.jar`, para ello en la terminal de [IntelliJ Idea](#) se debe utilizar el comando `mvnw package`. Este proceso puede demorar un par de minutos:

```
D:\account_ms>mvnw package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.misiontic:account_ms >-----
[INFO] Building account_ms 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ account_ms ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ account_ms ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ account_ms ---
```

Una vez finalice este proceso, se debe crear un archivo llamado [Dockerfile](#) (sin extensión) en la raíz del proyecto. Al hacerlo, la estructura del proyecto debe ser la siguiente:



Una vez se ha creado el *Dockerfile*, dentro de este se deben indicar las siguientes instrucciones:

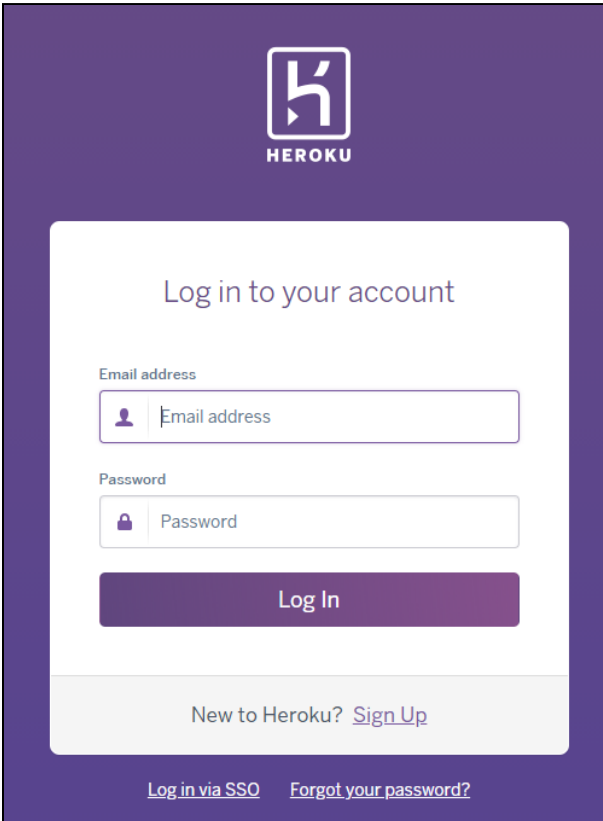
```
FROM openjdk:17-jdk-alpine
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
CMD gunicorn --bind 0.0.0.0:$PORT wsgi
```

De acuerdo con las características del proyecto, el *Dockerfile* asociado a este tiene unas instrucciones u otras. Estas instrucciones (también llamadas comandos), siguen la sintaxis definida por Docker en la documentación (<https://docs.docker.com/engine/reference/builder/>) para indicar los pasos que permitirán crear la imagen Docker del proyecto. Estas instrucciones indican, sin entrar en detalles, la imagen Docker base que se debe utilizar, la configuración del proyecto que se debe realizar, y el comando a utilizar para el despliegue de la aplicación.

Nota: de ser necesario, en el material de la clase se encuentra un archivo llamado *C4a.AP.06.account_ms.rar*, con los cambios realizados en el componente para su despliegue.

Despliegue en Heroku con Docker

Ahora que se ha hecho todo lo necesario, se puede realizar el despliegue. Para ello, primero se debe iniciar sesión en el CLI de Heroku, utilizando en la terminal el comando *heroku login*. Luego, al presionar la tecla espacio, se abrirá una ventana en el navegador que permitirá iniciar sesión:



HEROKU

Log in to your account

Email address

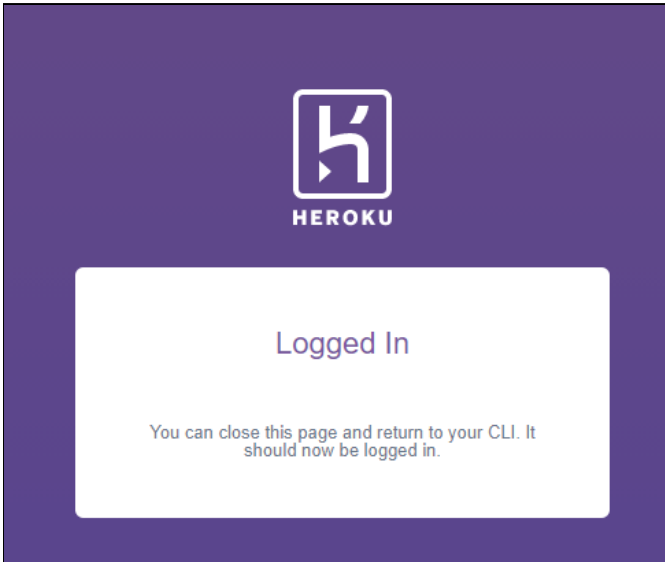
Password

Log In

New to Heroku? [Sign Up](#)

[Log in via SSO](#) [Forgot your password?](#)

Una vez se ingresan las credenciales y se pulsa el botón **Log In**, se muestra el siguiente mensaje en el navegador, y se actualiza la terminal donde se ingresó el comando:



HEROKU

Logged In

You can close this page and return to your CLI. It should now be logged in.

```
D:\account_ms>heroku login
» Warning: heroku update available from 7.56.1 to 7.59.0.
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/21bb738
Logging in... done
Logged in as misiontic2022@hotmail.com
```

Ahora se debe crear la imagen del proyecto, para esto Docker se debe estar ejecutando en el sistema (simplemente se debe abrir la aplicación). Así, con Docker corriendo se debe conectar la terminal con el servicio de Docker de Heroku, para esto se ejecuta comando *heroku container:login*:

```
D:\account_ms>heroku container:login
» Warning: heroku update available from 7.56.1 to 7.59.0.
Login Succeeded
```

Una vez se ha establecido la conexión, se debe crear la aplicación de Heroku donde se realizará el despliegue del microservicio *account_ms*. Para ello, se debe utilizar el comando *heroku create nombre-app* cambiando la palabra *nombre-app* por el nombre que se desea poner a la aplicación. En este caso se utilizará el comando con el nombre *mision-tic-account-ms*. Si el nombre es válido, se creará la aplicación correctamente:

```
D:\account_ms>heroku create mision-tic-account-ms
» Warning: heroku update available from 7.56.1 to 7.59.0.
Creating ● mision-tic-account-ms... done
https://mision-tic-account-ms.herokuapp.com/ | https://git.heroku.com/mision-tic-account-ms.git
```

Posteriormente, se debe crear la imagen del proyecto con el comando, *heroku container:push web --app nombre-app*, donde *nombre-app* es el nombre de la aplicación de Heroku creada anteriormente. Este comando puede tardar varios minutos en ejecutarse, cuando finalice se tendrá construida la imagen del proyecto:

```
D:\account_ms>heroku container:push web --app mision-tic-account-ms
» Warning: heroku update available from 7.56.1 to 7.59.0.
=== Building web (D:\account_ms\Dockerfile)
[+] Building 2.5s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 199B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/openjdk:17-jdk-alpine
=> [auth] library/openjdk:pull token for registry-1.docker.io
=> [internal] load build context
=> => transferring context: 23.13MB
=> CACHED [1/2] FROM docker.io/library/openjdk:17-jdk-alpine@sha256:4b6abae565492dbe9e7a894137c966a7485154238902f2f25e9dbd9784383d81
```

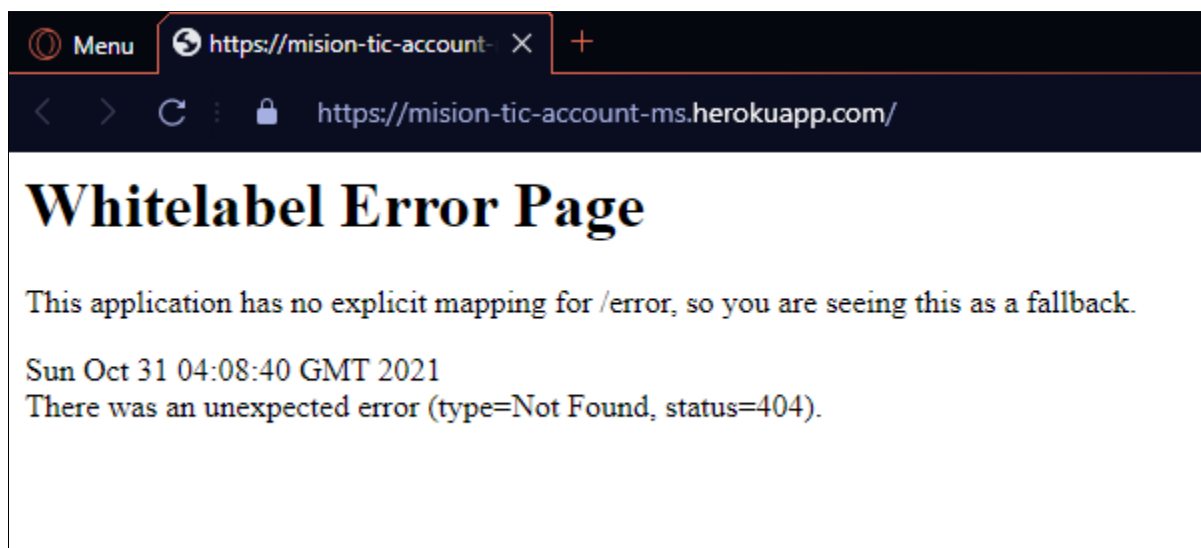
Una vez creada la imagen del proyecto, solo falta realizar el despliegue, para ello se utiliza el comando `heroku container:release web --app nombre-app`, donde `nombre-app` es el nombre de la aplicación de Heroku:

```
D:\account_ms>heroku container:release web --app mision-tic-account-ms
» Warning: heroku update available from 7.56.1 to 7.59.0.
Releasing images web to mision-tic-account-ms... done
```

Una vez realizado el despliegue del microservicio `account_ms`, se puede probar que las urls del servidor funcionan correctamente. Para esto se utilizará `Postman`, junto con la url de componente desplegado en heroku, cuyo formato, como ya se ha visto, es el siguiente:

<https://{nombre-app}.herokuapp.com/>

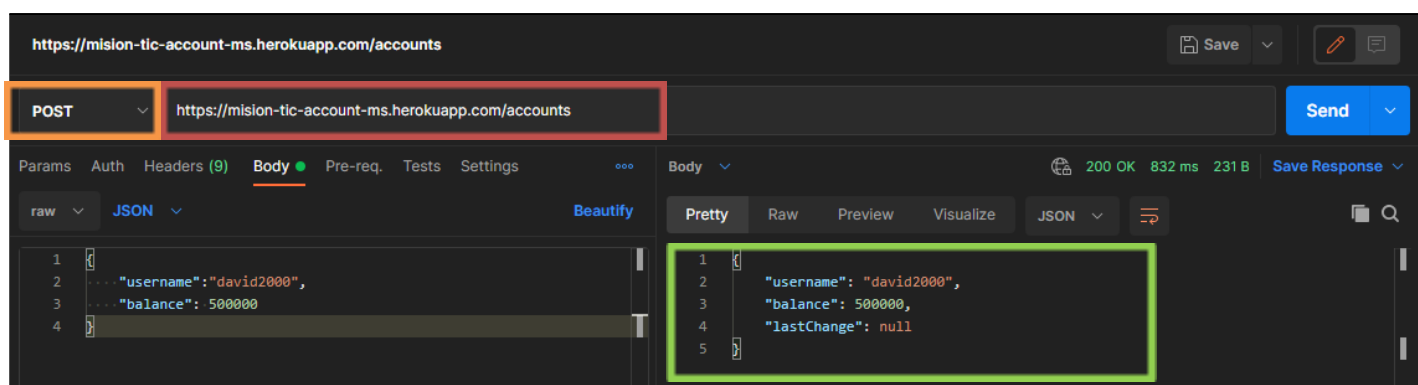
Si se accede a esta url, se podrá observar el siguiente mensaje:



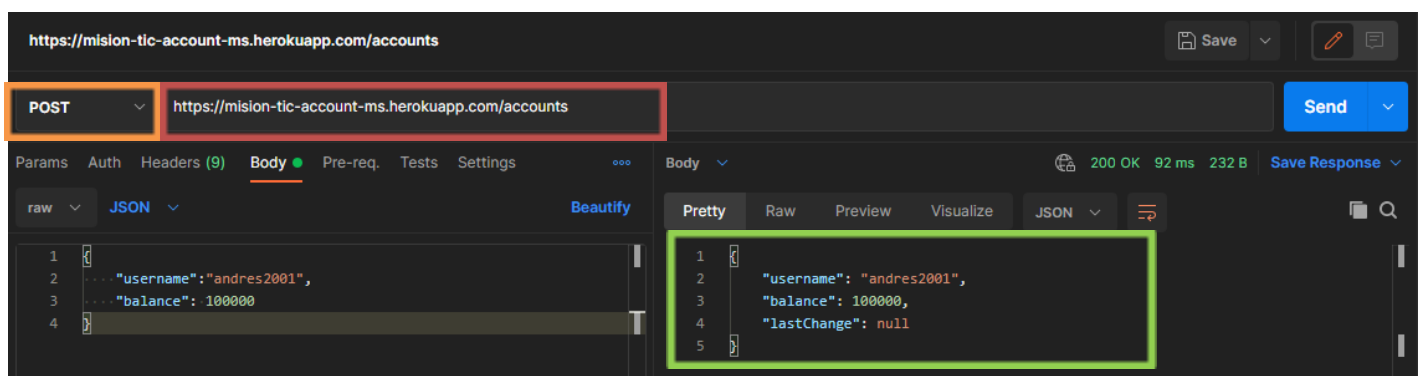
Lo cual indica que el despliegue se realizó satisfactoriamente. Toda funcionalidad se probará utilizando la anterior url, seguida por el sufijo de la funcionalidad a probar.

Prueba: Crear la cuenta de un usuario

La primera petición que se probará será aquella que se utiliza para crear la cuenta bancaria de un usuario del sistema. Para probarla, en **Postman** se realizará una petición **POST** a la url <https://{nombre-app}.herokuapp.com/accounts>, con un respectivo **Body** de tipo **JSON** que contenga la información del usuario. Esto deberá retornar la información de la cuenta del usuario recién creado:



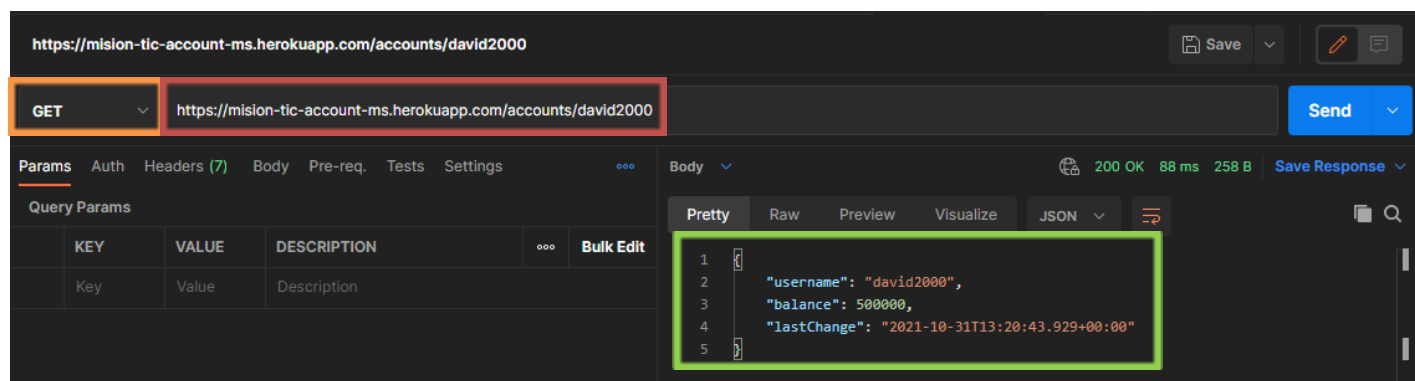
Para las siguientes pruebas es necesario, la construcción de la cuenta para otro usuario, esta petición debe lucir así:



Prueba: Consultar la cuenta de un usuario

La siguiente petición que se probará será aquella que se utiliza para obtener la información de la cuenta bancaria de un usuario del sistema. Para probarla, en **Postman** se realizará una petición **GET** a la url <https://{nombre-app}.herokuapp.com/accounts/david2000>. Esto deberá retornar la información de la

cuenta del usuario cuyo *username* es “david2000”:



https://mision-tic-account-ms.herokuapp.com/accounts/david2000

GET https://mision-tic-account-ms.herokuapp.com/accounts/david2000

Params Auth Headers (7) Body Pre-req. Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body

200 OK 88 ms 258 B

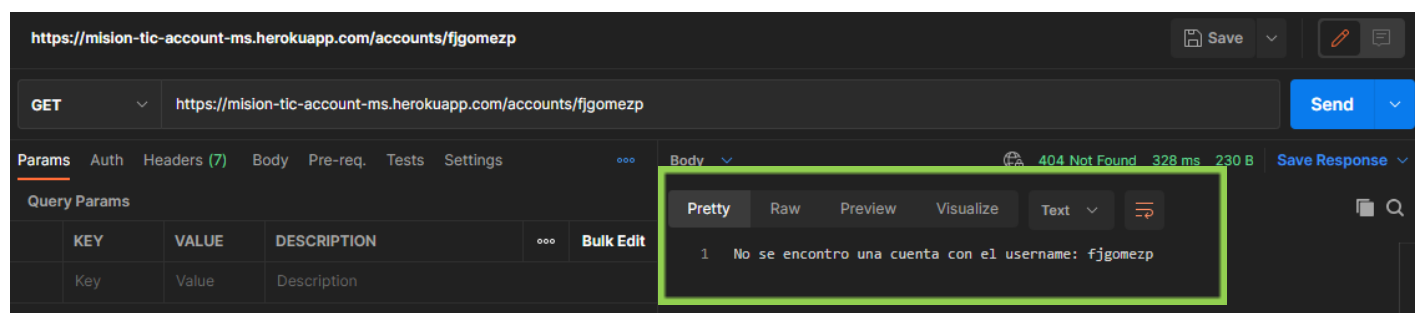
Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "username": "david2000",
3   "balance": 500000,
4   "lastChange": "2021-10-31T13:20:43.929+00:00"
5 }
```

Por otro lado, se puede comprobar que una de las excepciones personalizadas está funcionando correctamente, buscando una cuenta bancaria que no exista en el sistema. Por ejemplo, si se busca la cuenta asociada al username “fjgomezp” se debe obtener el siguiente resultado:



https://mision-tic-account-ms.herokuapp.com/accounts/fjgomezp

GET https://mision-tic-account-ms.herokuapp.com/accounts/fjgomezp

Params Auth Headers (7) Body Pre-req. Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body

404 Not Found 328 ms 230 B

Save Response

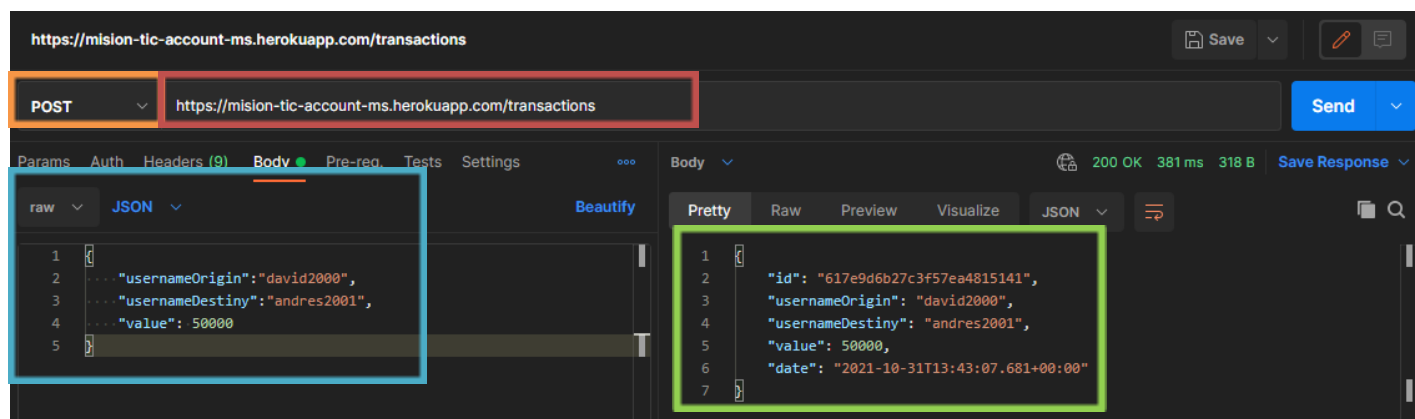
Pretty Raw Preview Visualize Text

```

1 No se encontro una cuenta con el username: fjgomezp
```

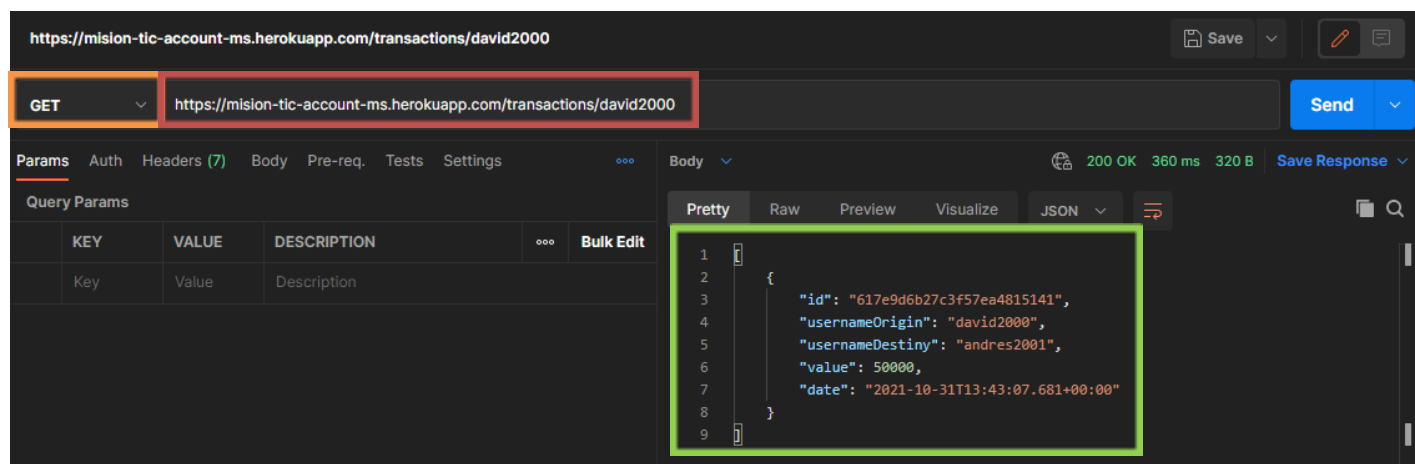
Prueba: Realizar una transacción

La siguiente petición que se probará es aquella con la que se realiza una transacción entre dos cuentas. Para ello, en *Postman* se debe realizar una petición *POST* a la url <https://{nombre-app}.herokuapp.com/transactions>, incluyendo un *Body* de tipo *JSON* con la información necesaria para realizar la petición. Al realizar la petición, se debe obtener el siguiente resultado:



Prueba: Consultar las transacciones de un usuario

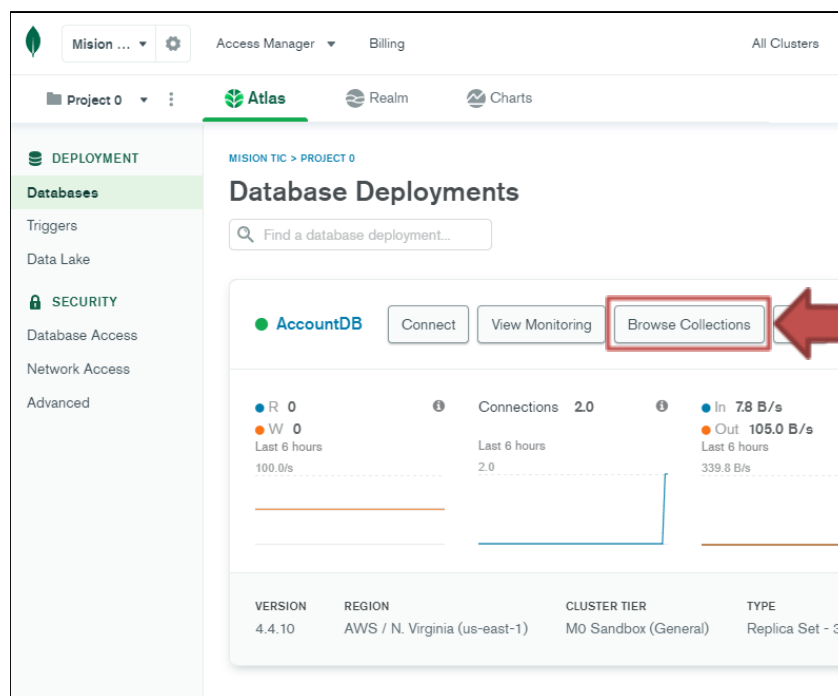
La última petición que se probará es aquella con la que se consulta todas las transacciones de un usuario, ya sea que este sea el que envía o recibe el dinero en la transacción. Para ello, en [Postman](#) se debe realizar una petición [GET](#) a la url <https://{nombre-app}.herokuapp.com/transactions/david2000>. Esto deberá retornar las transacciones del usuario cuyo *username* es "david2000".



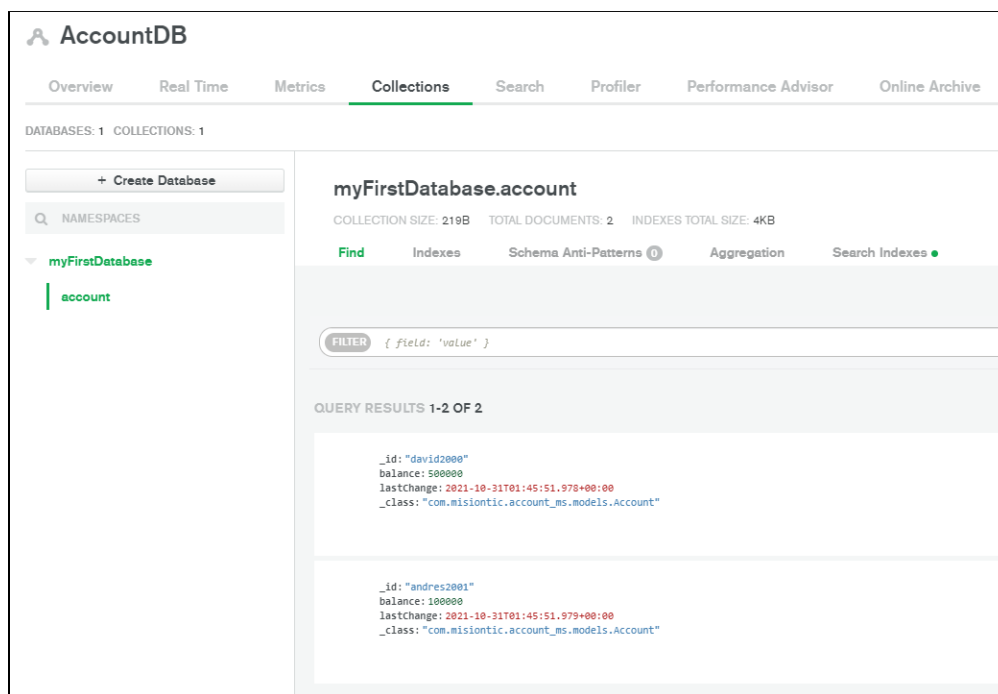
Una vez se ha realizado la petición, se ha finalizado el desarrollo y las pruebas del microservicio [account_ms..](#)

Revisión de cambios en la Base de Datos

Desde el panel de control de Atlas se puede acceder a la estructura de la base de datos de MongoDB, para esto únicamente se debe hacer clic en la opción ["Browse Collections"](#) :



Una vez realizada la petición que permite crear una cuenta, se puede evidenciar el cambio en la base de datos. Como es de esperarse, se las dos cuentas fueron creadas:



Asimismo, se puede evidenciar que la transacción realizada anteriormente se guardó correctamente:

MISSION TIC > PROJECT 0 > DATABASES

AccountDB VERSION 4.4.10 REGION AWS N. Virginia (us-east-1)

Overview Real Time Metrics **Collections** Search Profiler Performance Advisor Online Archive Command Line Tools

DATABASES: 1 COLLECTIONS: 2 VISUALIZE YOUR DATA REFRESH

+ Create Database

Q NAMESPACES

▼ **myFirstDatabase**

account

transaction

myFirstDatabase.transaction

COLLECTION SIZE: 165B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 20KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

INSERT DOCUMENT

FILTER { field: "value" } OPTIONS Apply Reset

QUERY RESULTS 1-1 OF 1

```

_id: ObjectId("617e9d6b27c3f57ea4815141")
usernameOrigin: "david2000"
usernameDestiny: "andres2001"
value: 50000
date: 2021-10-31T13:43:07.681+00:00
_class: "com.misiontic.account_ms.models.Transaction"

```