



El futuro digital
es de todos

MinTIC

```
te( "name" );  
"type" );
```

```
if ( type == "sprite" )  
  
std::string item_name = item->Attribute( "name" );  
std::string spritename = item->Attribute( "spritename" );  
float x = boost::lexical_cast<float>( item->Attribute( "x" ) );  
float y = boost::lexical_cast<float>( item->Attribute( "y" ) );  
float offset = boost::lexical_cast<float>( item->Attribute( "offset" ) );  
  
SpriteDescList::iterator sp = sprite_descs.begin();  
for( ; sp != sprite_descs.end(); ++sp )  
    if ( sp->name_ == spritename )  
        break;
```

Ciclo 3:

Desarrollo de Software



**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) + 314 5000
uec_ibog@unaleduco

Componente Lógico

(Despliegue y Pruebas)

Actividad Práctica

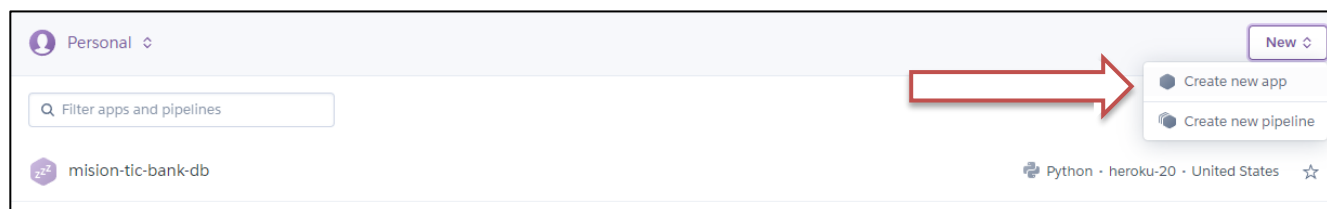
Hasta este punto el componente de backend (*bank_be*) se ha ejecutado únicamente en un entorno de desarrollo local (el equipo en el cual se ha desarrollado el componente). Esto presenta una gran limitación a nivel de acceso, ya que solamente las personas que tengan acceso al equipo pueden acceder al componente. Para eliminar esta limitación se debe ejecutar el componente en un entorno remoto en el cual todos los clientes tengan acceso, a este proceso de ejecución en un entorno remoto se le conoce comúnmente como *despliegue remoto*.

Los entornos sobre los cuales se realizan los despliegues remotos son computadores especializados que se conocen comúnmente como *servidores*. Si bien se puede crear un servidor desde cero para realizar un proceso de despliegue, en pocos casos es una buena opción, ya que existen plataformas conocidas como *PaaS* (por sus siglas en Inglés Platform as a Service) que ofrecen las herramientas necesarias para realizar procesos de construcción y despliegue de aplicaciones, de una manera más rápida, y sin depender de las configuraciones específicas que se deberían realizar en los servidores. En estas, únicamente se debe indicar el código fuente del componente a desplegar y algunas instrucciones para llevar a cabo dicho despliegue. Una de las *PaaS* más comunes es Heroku, y al igual que en la capa de datos, esta se utilizará para realizar el despliegue del componente de back-end.

Creación APP Heroku

Para llevar a cabo el proceso de despliegue se debe crear una aplicación de *Heroku*, en dicha aplicación se realizará el proceso de despliegue del componente de *back-end*.

Para crear la aplicación dentro del *Dashboard* de *Heroku* (con la cuenta creada previamente), se selecciona el botón *Create new app*:



Luego se debe nombrar la App con un nombre descriptivo (en este caso *mision-tic-bank-be*) y se crea la app:

Create New App

App name

mision-tic-bank-be

✓

mision-tic-bank-be is available

Choose a region

United States

⌵

Add to pipeline...

Create app

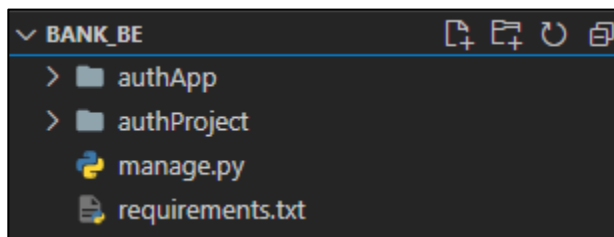
Con esto se tendrá una App con la cual se podrá realizar el proceso de Despliegue.

Cambios en Código

Para que Heroku pueda realizar el proceso de despliegue del componente es necesario realizar algunos cambios sobre el código, estos cambios son establecidos por [Heroku](#) y son estándares para cualquier proyecto de [Django](#) o [DjangoREST](#).

Requeriments.txt

En raíz del proyecto se debe crear un archivo llamado [requirements.txt](#):



El objetivo de este archivo es incluir todos y cada uno de los paquetes (incluido su versión) necesarios para la ejecución y despliegue del componente, el contenido de este archivo será el siguiente:

```
django==3.2.7
djangorestframework==3.12.4
djangorestframework-simplejwt==4.8.0
psycopg2==2.9.1
gunicorn==20.1.0
django-heroku==0.3.1
```

Los primeros 4 paquetes fueron utilizados en el desarrollo del componente, mientras que los dos últimos son necesarios en el proceso de despliegue.

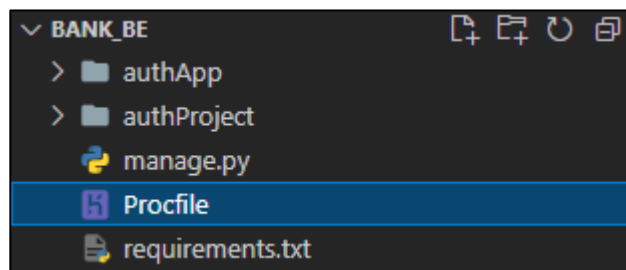
Settings.py

El paquete [django-heroku](#) realiza algunas operaciones que permiten a Heroku realizar el despliegue. Para hacer uso de este paquete se deben incluir las siguientes dos líneas de código al final del archivo [authProject/settings.py](#):

```
import django_heroku
django_heroku.settings(locals())
```

Procfile

En raíz del proyecto se debe crear un archivo llamado [Procfile](#) (sin extensión):

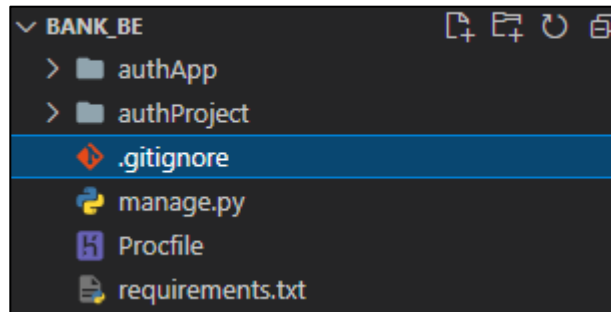


Dentro de este archivo se deben establecer las instrucciones que permiten ejecutar el componente, en este caso solamente se debe inicializar el servicio, por lo cual el contenido del archivo [Procfile](#) será únicamente la siguiente línea:

```
web: gunicorn authProject.wsgi
```

Git Ignore

En raíz del proyecto se debe crear un archivo llamado `.gitignore`:



Durante el desarrollo del componente se usaron algunos elementos como el [entorno virtual](#) y es posible que algunos de estos archivos permanezcan en el código fuente. Ya que estos archivos no serán utilizados en el proceso de despliegue, con ayuda del archivo `.gitignore` se omitirán. Para lograr esto, se debe incluir la siguiente línea:

```
/env/
```

Despliegue del Componente

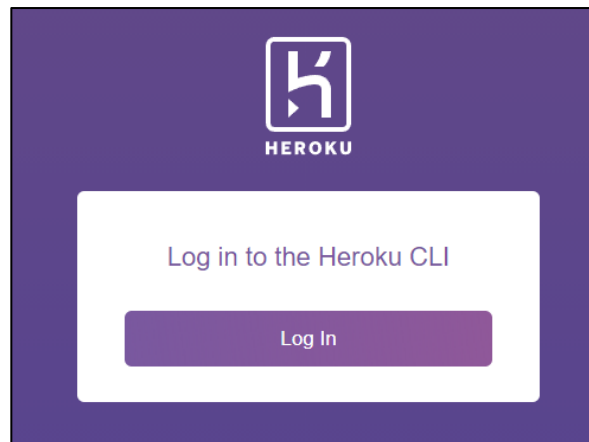
Una vez se han realizado los cambios necesarios en el proyecto, se puede llevar a cabo el proceso de [despliegue](#). Para realizar dicho proceso, se debe tener en cuenta que [Heroku](#) funciona usando un sistema basado en [git](#), por lo cual, a grosso modo el proceso consistirá en realizar un [commit](#) y subir este [commit](#) a un [repositorio](#) remoto administrado por [Heroku](#). A continuación, se detallará el paso a paso para llevar a cabo el despliegue del componente.

Heroku Login:

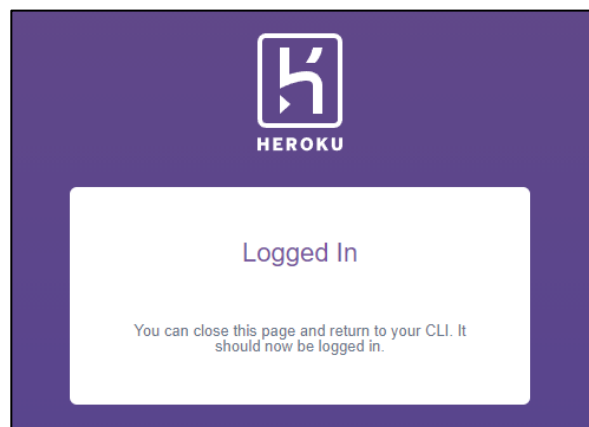
Para poder establecer contacto con Heroku, se debe iniciar sesión con ayuda de su CLI (previamente instalado), esto se logra ejecutando el comando el siguiente comando en un `cmd`, `bash` o `terminal` y oprimiendo la tecla espacio:

heroku login

Esto abrirá una ventana en el navegador que permitirá el inicio de sesión:



Una vez *inciada la sesión*, debe salir el siguiente mensaje en el navegador y la terminal se debe actualizar:



Todos los comandos que aparezcan en la guía se ejecutarán en esta consola, por ello es importante que durante el proceso no se cierre.

Crear repositorio

Dado que *Heroku* maneja el código de despliegue como un repositorio de git, se debe *crear* o inicializar dicho repositorio. Esto se logra ejecutando el siguiente comando en la raíz del código fuente:

```
git init
```

Si el proyecto ya ha sido inicializado como repositorio previamente, se puede omitir este paso.

Agregar Remoto

El código fuente del proyecto será subido a un repositorio remoto administrado por Heroku, para agregar este repositorio remoto al repositorio local se debe ejecutar el siguiente comando:

```
heroku git:remote -a name-app
```

Se debe tener en cuenta el nombre de la aplicación creada en Heroku.

Realizar Commit

Se debe crear un commit que incluya todos los archivos referentes al estado actual del proyecto, incluidos los últimos cambios realizados, esto se logra con los siguientes comandos:

```
git add .  
git commit -am "deployment changes"
```

Realizar Despliegue

En este punto para realizar el despliegue únicamente bastará con subir el último commit al repositorio remoto de heroku, esto se logra con el siguiente comando:

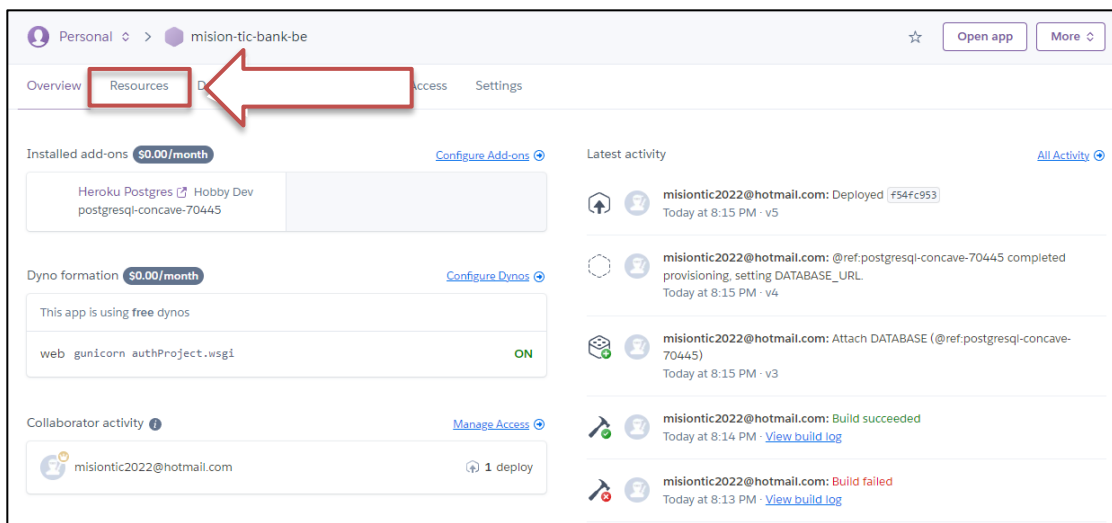
```
git push heroku master
```

Al ejecutar este comando se mostrará el estado del despliegue hasta finalizar.

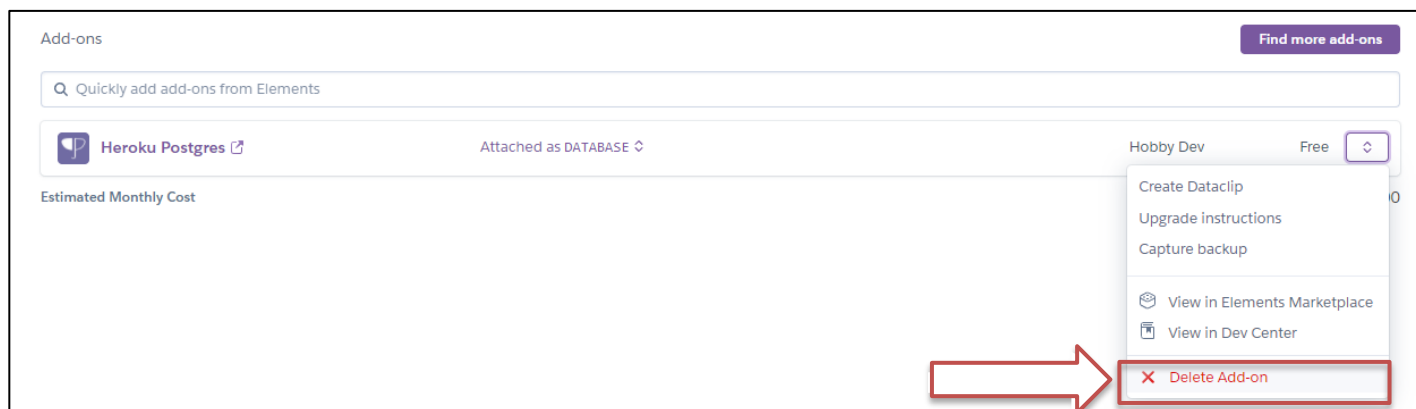
Eliminar Recurso Database:

Por defecto Heroku detecta que la aplicación de Django REST está trabajando con una base de datos de PostgreSQL y crea su propio recurso, esto genera un conflicto ya que, en este caso, se está usando un componente de base de datos alojado en otra aplicación (también de Heroku). Para solventar este conflicto se debe eliminar el recurso de PostgreSQL creado por Heroku en la aplicación en la que fue desplegado el componente de back--end.

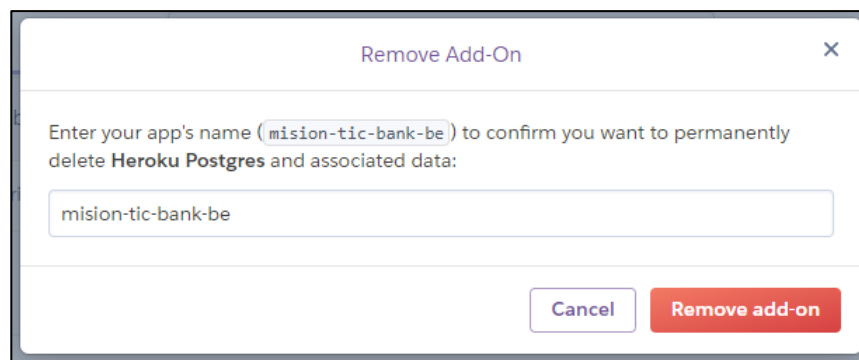
Para eliminar el recurso, en el *Dashboard* de la aplicación de back-end se debe acceder a *Resources*:



En la parte de opciones, se debe seleccionar Delete Add-on:



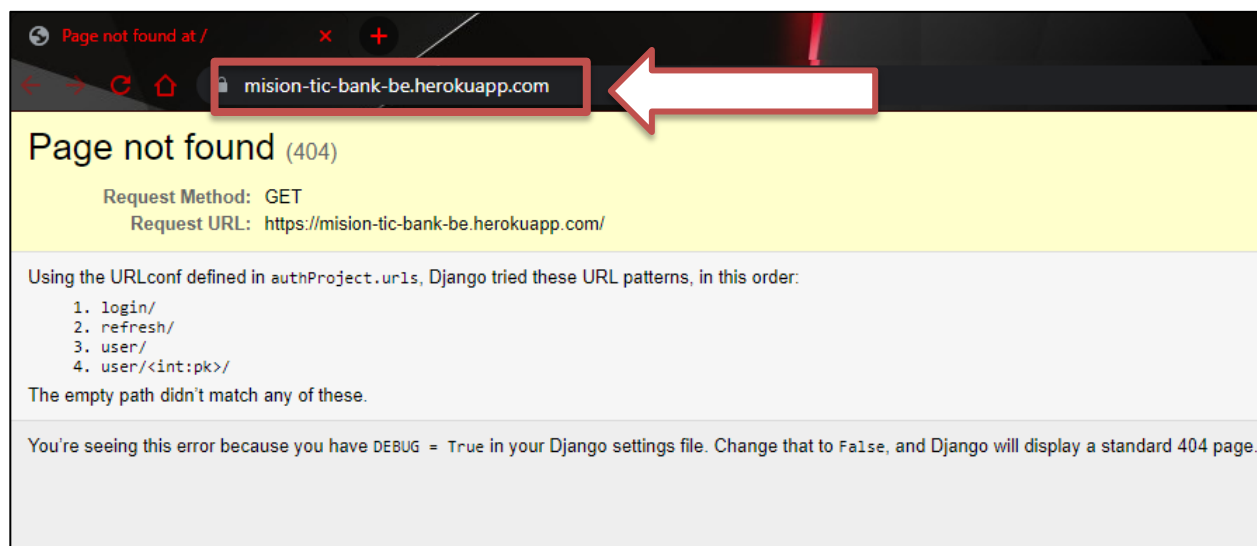
Y luego confirmar la eliminación:



Con esto se conseguirá solventar el conflicto.

Obtener URL del componente

Finalmente, para conocer la [URL](#) de acceso al componente, en el [Dashboard](#) de la aplicación de [Heroku](#), se selecciona [Open App](#) y se redirige a la [URL](#) del componente:



Pruebas del componente lógico

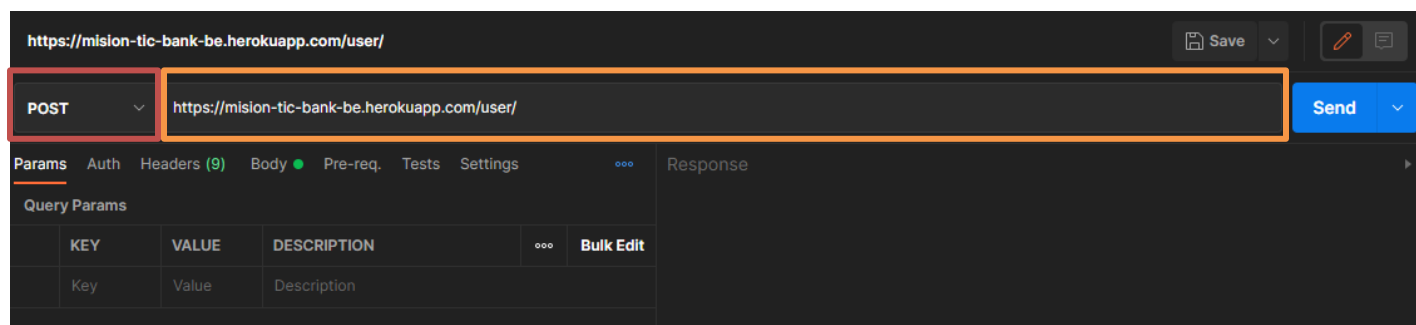
Una vez realizado el despliegue del componente lógico [bank_be](#), se puede comprobar que las URLs del servidor funcionan correctamente. Para esto se utilizará [Postman](#), junto con la URL de componente desplegado en Heroku, cuyo formato es el siguiente:

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

Toda funcionalidad se probará con la URL de la aplicación de Heroku, seguida por el sufijo de la funcionalidad a probar.

Funcionalidad: Registrar usuario

La primera petición que se comprobará será la correspondiente a la creación del usuario. Para probarla, se debe ingresar a la URL <https://{nombre-app}.herokuapp.com/user/>, en el espacio que se encuentra a la izquierda del botón [Send](#), y se debe indicar que el tipo de petición a realizar es [POST](#) en el selector de opciones que se encuentra a la izquierda del espacio donde se ingresa la URL.



https://mision-tic-bank-be.herokuapp.com/user/

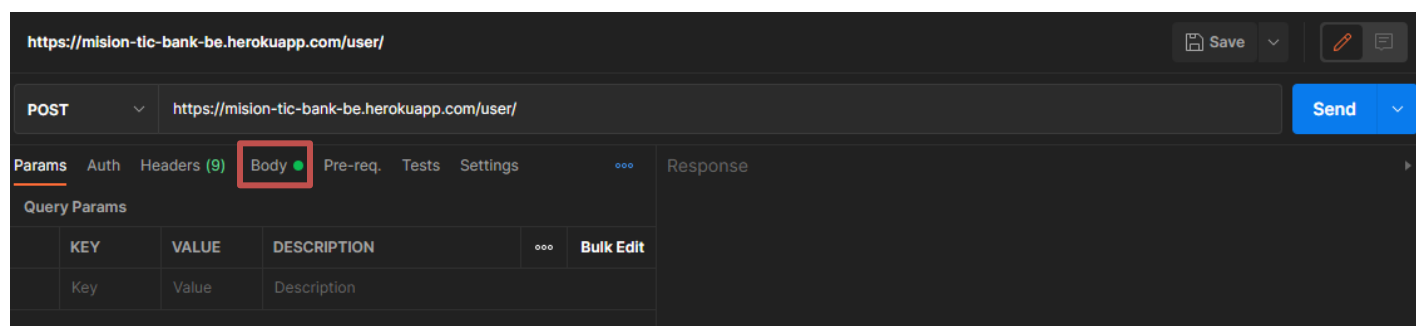
POST ▼ https://mision-tic-bank-be.herokuapp.com/user/ Send ▼

Params Auth Headers (9) Body ● Pre-req. Tests Settings ⋮ Response

Query Params

	KEY	VALUE	DESCRIPTION	⋮	Bulk Edit
	Key	Value	Description		

Posteriormente, en la sección izquierda, bajo el espacio para el ingreso de la URL, se debe seleccionar la opción **Body**:



https://mision-tic-bank-be.herokuapp.com/user/

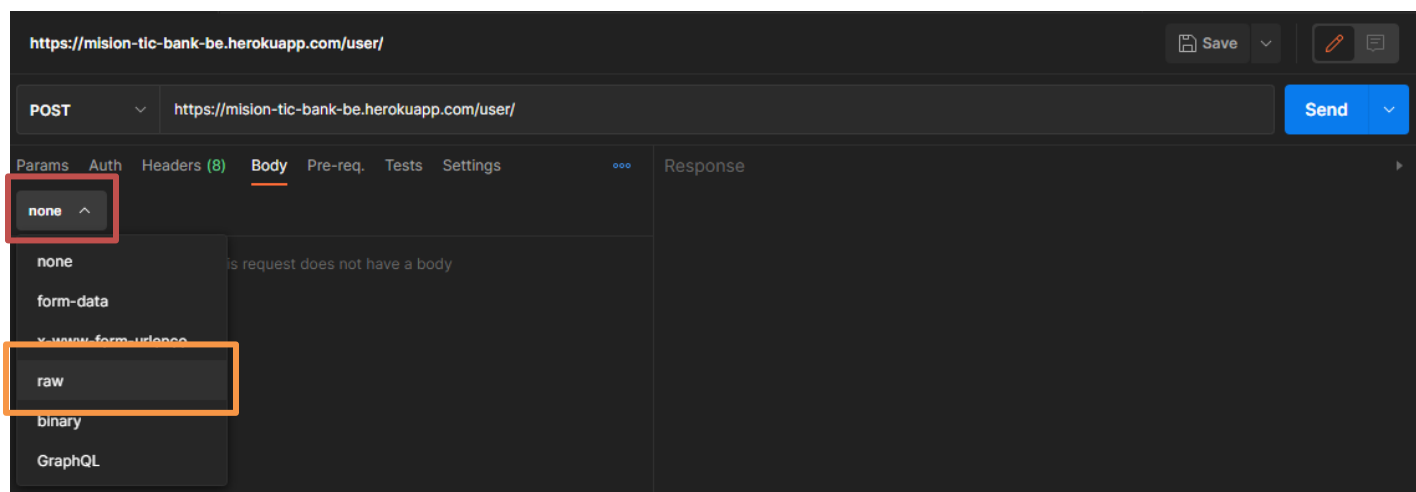
POST ▼ https://mision-tic-bank-be.herokuapp.com/user/ Send ▼

Params Auth Headers (9) **Body** ● Pre-req. Tests Settings ⋮ Response

Query Params

	KEY	VALUE	DESCRIPTION	⋮	Bulk Edit
	Key	Value	Description		

Luego, se debe seleccionar el formato que tendrá la información a enviar en la petición, para ello, se debe seleccionar la opción **raw**, y posteriormente la opción **JSON**:



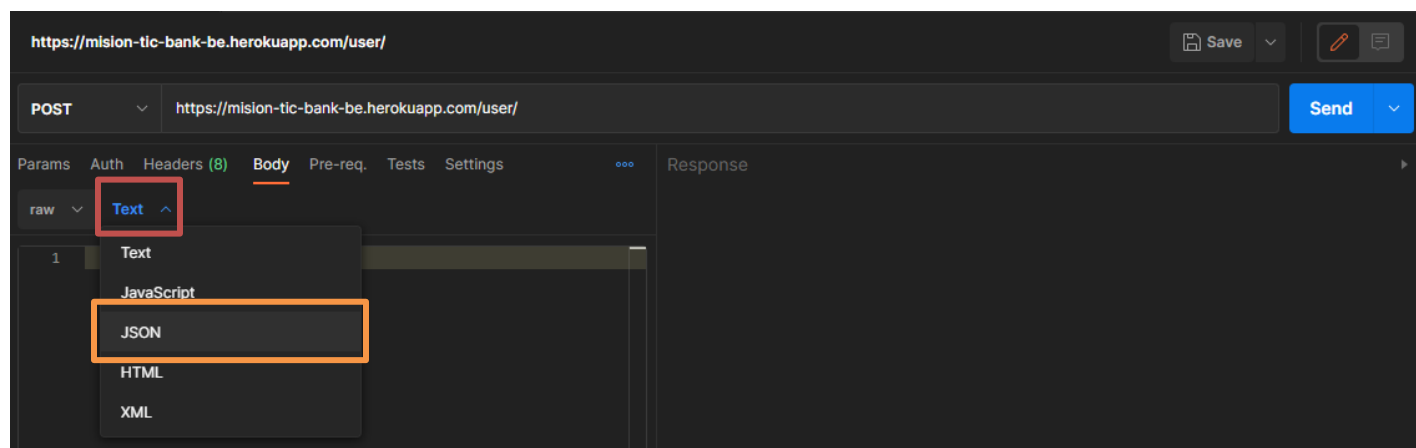
https://mision-tic-bank-be.herokuapp.com/user/

POST ▼ https://mision-tic-bank-be.herokuapp.com/user/ Send ▼

Params Auth Headers (8) **Body** ● Pre-req. Tests Settings ⋮ Response

none ^

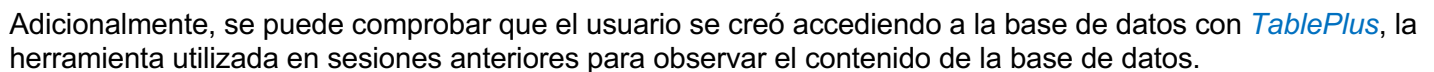
- none
- form-data
- x-www-form-urlencoded
- raw**
- binary
- GraphQL



Allí, en el espacio de la sección izquierda se debe ingresar el **JSON** que contiene la información del usuario que se va a registrar en el sistema. Un **JSON** válido de ejemplo que se puede utilizar es el siguiente:

```
{
  "username": "fjgomezpe",
  "password": "contraseña",
  "name": "francisco",
  "email": "fjgomezpe@misionTIC.com",
  "account": {
    "lastChangeDate": "2021-09-23T10:25:43.511Z",
    "balance": 20000,
    "isActive": true
  }
}
```

Una vez se ingresa y se pulsa el botón **Send**, en la sección derecha se obtiene la respuesta del servidor. Si el componente lógico se desarrolló correctamente, la respuesta debe ser similar a la siguiente:



https://mision-tic-bank-be.herokuapp.com/login/

Save

POST

https://mision-tic-bank-be.herokuapp.com/login/

Send

Params

Auth

Headers (9)

Body

Pre-req.

Tests

Settings

Query Params

KEY	VALUE	DESCRIPTION		Bulk Edit
Key	Value	Description		

Response

```
{
  "username": "fjgomezpe",
  "password": "contraseña"
}
```

The screenshot shows a REST client interface. The top bar displays the URL `https://mision-tic-bank-be.herokuapp.com/login/`. Below the URL bar, the request method is set to **POST**. The request body is in JSON format, containing the following data:

```
{
  "username": "fjgomezpe",
  "password": "contraseña"
}
```

The response status is **200 OK** with a response time of **564 ms**. The response body is also in JSON format, containing the following data:

```
{
  "refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1IjoicmVmcVzaCisImV4ciI6MTYzMjYyMTA5MSwianRpIjoizWZhODc1NTZkMzhkNDRIOWJkMDNlMTFjOTQ1MTg1NTAiLCJ1c2VyX2lki3fQ.KqeMcKk3kCbXGoPyMYX3OCm3y8AbBa1SW00m8SBWYf4",
  "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1IjoicmVmcVzaCisImV4ciI6MTYzMjYyMTA5MSwianRpIjoizWZhODc1NTZkMzhkNDRIOWJkMDNlMTFjOTQ1MTg1NTAiLCJ1c2VyX2lki3fQ.BgePOhuLx30y19TqwV8p5nq9gb2Tfh1zgEEN9NZ5g4k"
}
```

La próxima petición que se comprobará será la correspondiente a la actualización del access token. Para probarla, se debe ingresar a la URL <https://{nombre-app}.herokuapp.com/refresh/>, y se debe indicar *POST* como el tipo de petición a realizar:

https://mision-tic-bank-be.herokuapp.com/refresh/

Save

POST

https://mision-tic-bank-be.herokuapp.com/refresh/

Send

Params

Auth

Headers (9)

Body

Pre-req.

Tests

Settings

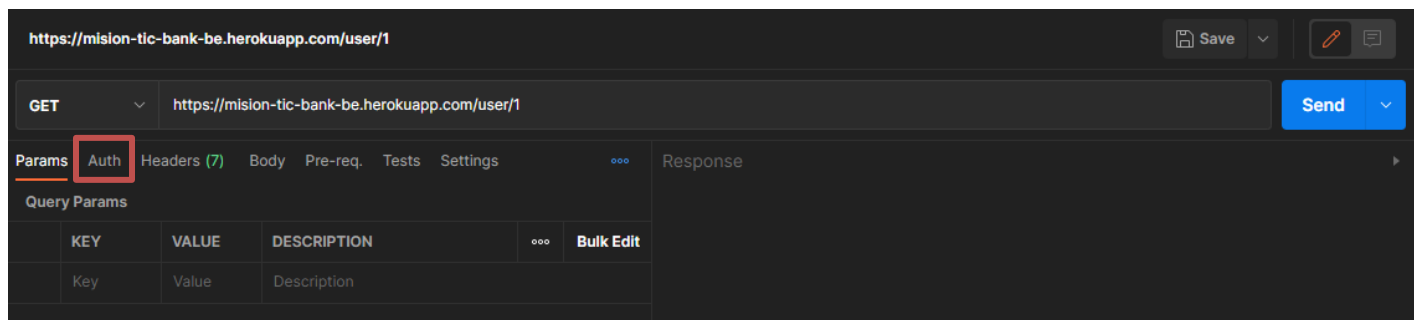
Query Params

KEY	VALUE	DESCRIPTION		Bulk Edit
Key	Value	Description		

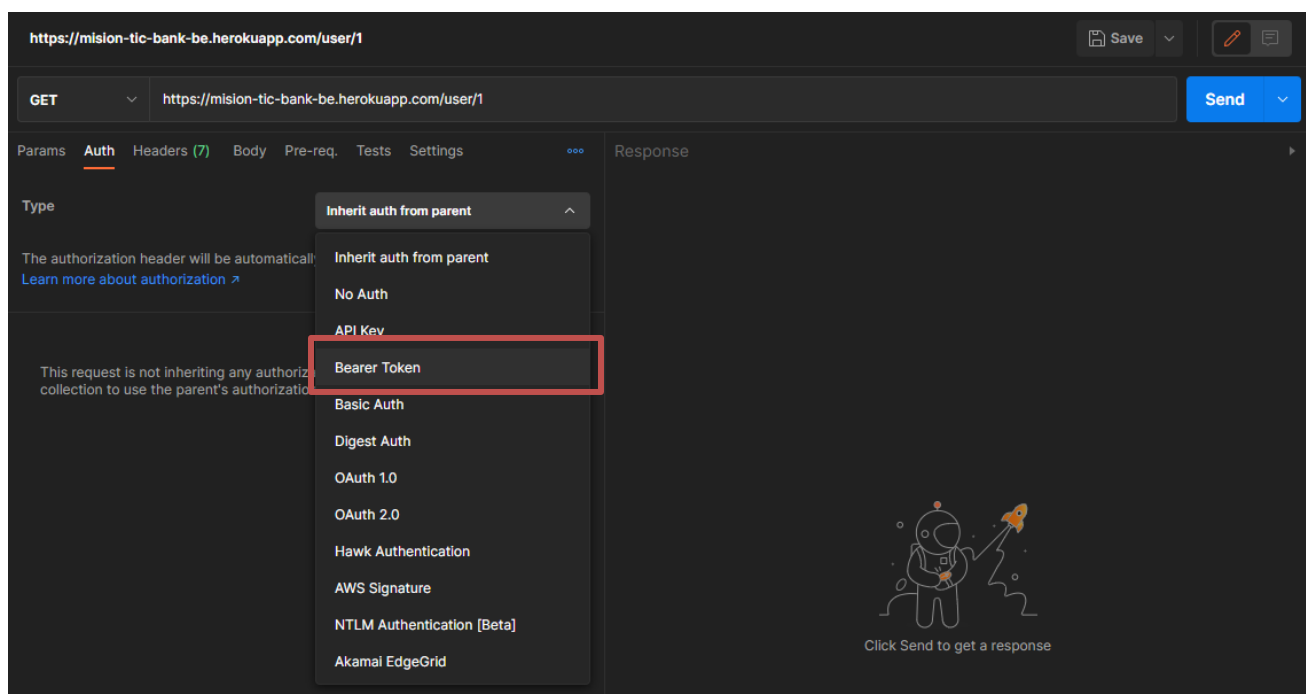
Facultad de
INGENIERÍA
Cede Bogotá

A screenshot of a REST client interface. The top bar shows the URL 'https://mision-tic-bank-be.herokuapp.com/refresh/'. Below it, the method is 'POST' and the URL is repeated. The 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response is a JSON object with a single key 'access' containing a long alphanumeric string. The status bar at the bottom indicates a 200 OK response with 332 ms latency and 507 B of data.

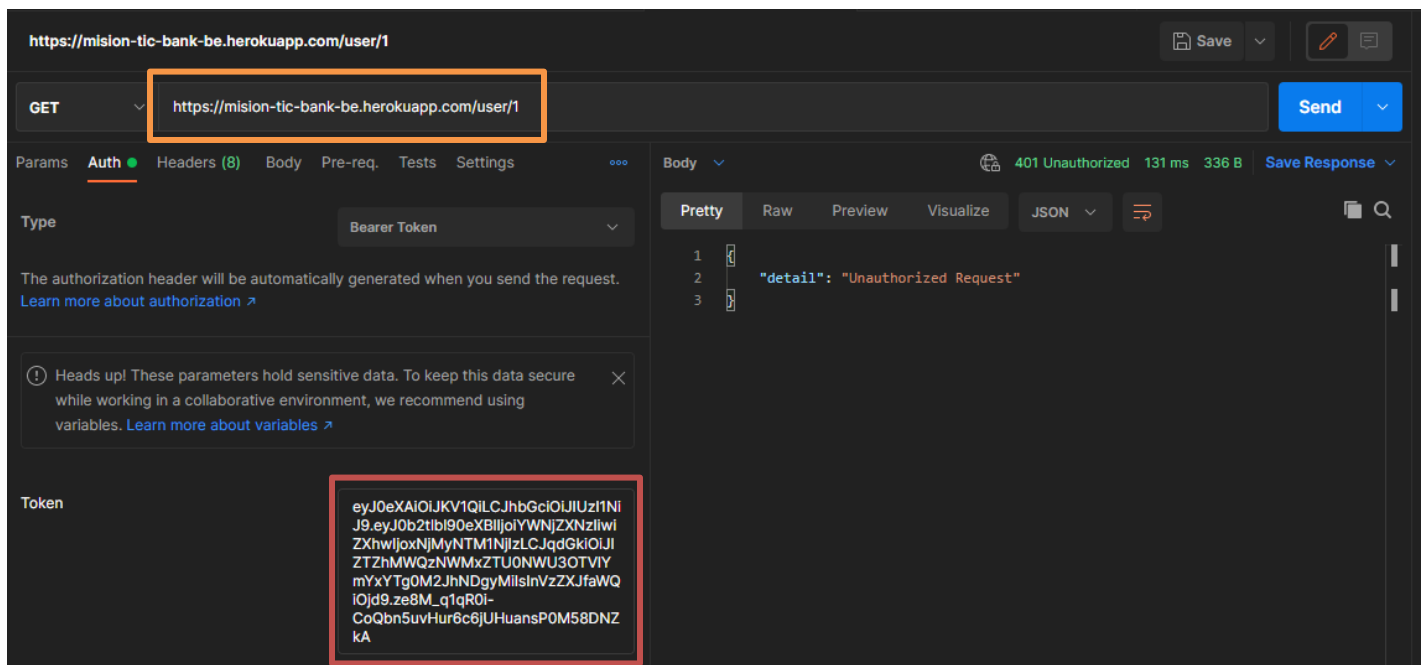
[illegible]



Luego, se debe seleccionar el tipo de *authorization header* que tendrá la petición HTTP:



Finalmente, se ingresa el access token en el espacio destinado para ello, y se pulsa el botón *Send*. Si el componente lógico se desarrolló correctamente, la respuesta puede ser alguna de las siguientes:



Nota: de ser necesario, en el material de la clase se encuentra un archivo *C3.AP.11. bank_be.rar*, con todos los avances en el desarrollo del componente realizado en esta guía.