



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 e 45-67  
Bloque 85 piso 1



(57) + 316 5000  
uec\_ibog@unaleduco

# Componente de Presentación

## (Despliegue y Pruebas)

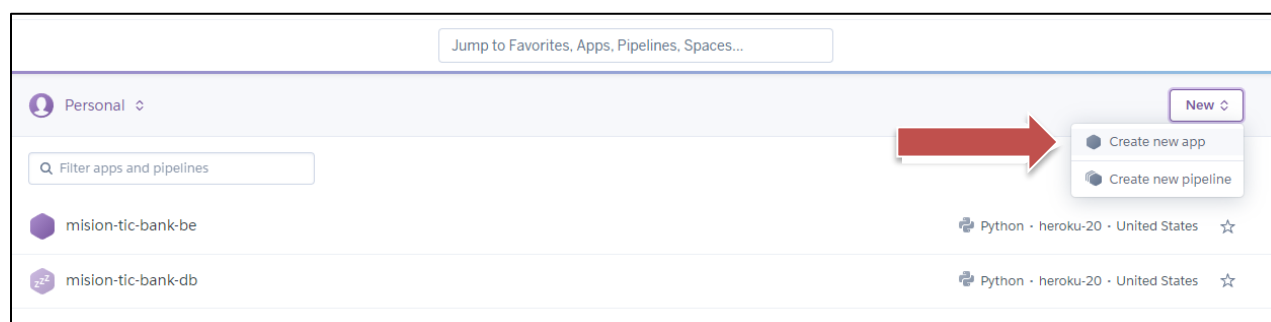
### Actividad Práctica

Una vez se ha finalizado el desarrollo del *componente* de *front-end* (*bank\_fe*), se debe realizar su despliegue remoto, para que todos los usuarios tengan acceso a este. Al igual que el componente de la capa lógica, el despliegue del componente de la capa de presentación se realizará a través de la plataforma *Heroku*, su funcionamiento será similar salvo por dos detalles, que, si bien están presentes en la capa lógica, se hacen mucho más notorios en la capa de presentación. Esto se debe a que el acceso al componente de *front-end* se realiza a través de un navegador, y este con el objetivo de brindar seguridad a sus usuarios, presta especial atención a los mecanismos de verificación sobre los sitios web. Los dos mecanismos de seguridad más importantes son la conexión *HTTPS* y sus *certificados*. A groso modo, una conexión *HTTPS* cumple el mismo objetivo de una conexión *HTTP* (es decir, acceder a recursos de un servidor desde un cliente), pero con una capa de seguridad adicional, en dicha capa se realizan procesos de *cifrado* y *verificación* del sitio. Para lo segundo utiliza el concepto de *certificados*, los cuales acreditan a un sitio web como seguro, estos certificados son expedidos por *entidades especializadas*. Si bien los conceptos de seguridad *HTTPS* y los certificados son un poco avanzados, en la práctica no representan trabajo adicional ya que *Heroku* se encarga de todo este proceso.

### Crear App Heroku

Dado que el despliegue se realizará a través de la plataforma *Heroku*, lo primero que se debe realizar es crear una aplicación o *app* para el componente de frontend.

Primero se debe acceder a la plataforma *Heroku*, e iniciar el proceso de creación de una nueva aplicación:



Ahora se debe ingresar un *nombre descriptivo* para la aplicación y luego crear la aplicación:

Con los pasos anteriores se ha creado una aplicación, la cual servirá para realizar el proceso de despliegue del componente de *front-end*, es muy importante recordar el *nombre de la aplicación*.

## Editar Código

Para llevar a cabo el despliegue del componente *bank\_fe* con *Heroku*, es necesario realizar algunos cambios sobre el código, estos cambios permitirán crear el *servidor* sobre el cual *Heroku* realizará el despliegue del componente.

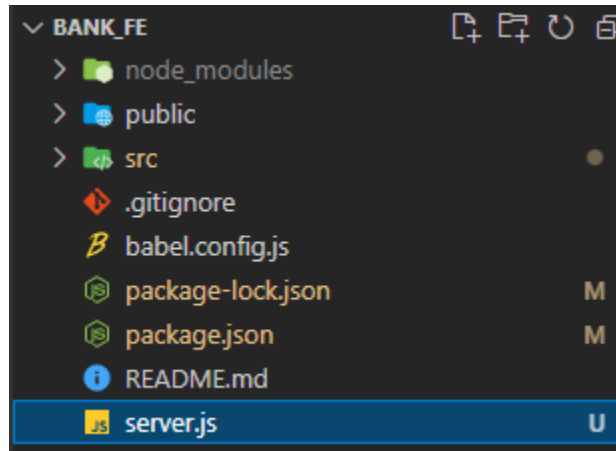
## Instalar Paquetes

Para poder crear el servidor que usará *Heroku* para llevar a cabo el despliegue, es necesario instalar algunos *paquetes* de *NodeJs* en el proyecto. Estos paquetes son *express* y *server-stactic*, y se instalan ejecutando el siguiente comando en una *consola* ubicada en la *raíz del proyecto*

```
npm install express serve-static --save
```

## Crear Server

El servidor se creará a través de un *script*, por cual se debe crear un archivo llamado *server.js* el cual estará en la raíz del proyecto, la estructura del proyecto debe lucir así:



El contenido del archivo *server.js* será el siguiente:

```
const express = require('express')
const serveStatic = require('serve-static')
const path = require('path')

const app = express()

//here we are configuring dist to serve app files
app.use('/', serveStatic(path.join(__dirname, '/dist')))

// this * route is to serve project on different page routes except root `/'
app.get(/.*/, function (req, res) {
  res.sendFile(path.join(__dirname, '/dist/index.html'))
})

const port = process.env.PORT || 8080
app.listen(port)
console.log(`app is listening on port: ${port}`)
```

Este código es sencillo, básicamente con ayuda de los *paquetes* instalados se *crea* un *servidor* a partir de *algunos* parámetros, como el puerto y la raíz del proyecto.

## Ejecutar Script

Ahora que se tiene el script para poder crear el servidor, se debe indicar al proyecto que al iniciarse ejecute el script, esto se logra modificando la sección de `scripts` del archivo `package.json` del proyecto. Allí, se debe añadir la instrucción `"start": "node server.js"`, de tal manera que luzca así:

```
"scripts": {  
  "serve": "vue-cli-service serve",  
  "build": "vue-cli-service build",  
  "start": "node server.js"  
},
```

Al momento de realizar el despliegue Heroku ejecutará este comando y se creará el servidor.

## Despliegue

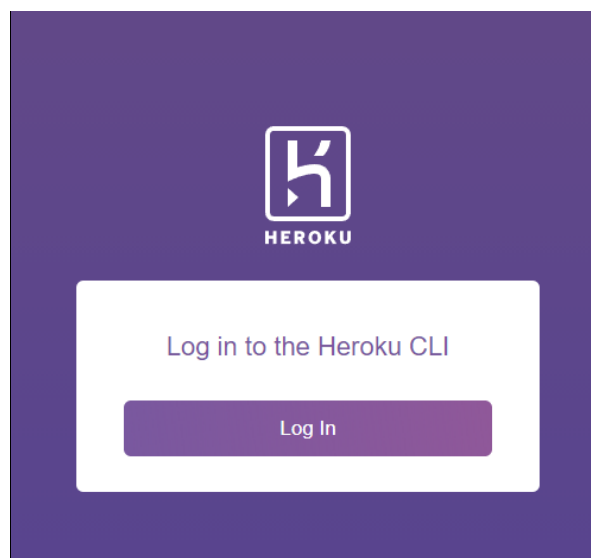
Una vez se ha modificado el código se puede realizar el proceso de despliegue con [Heroku](#), esto se realiza ejecutando algunos comandos desde una [consola](#) en la raíz del proyecto, estos comandos básicamente permiten conectarse [con](#) Heroku a través de su [CLI](#) (ya usado en guías anteriores) y con ayuda de [git](#), indicar cual es la versión del proyecto que se desea desplegar.

## Conectarse a Heroku

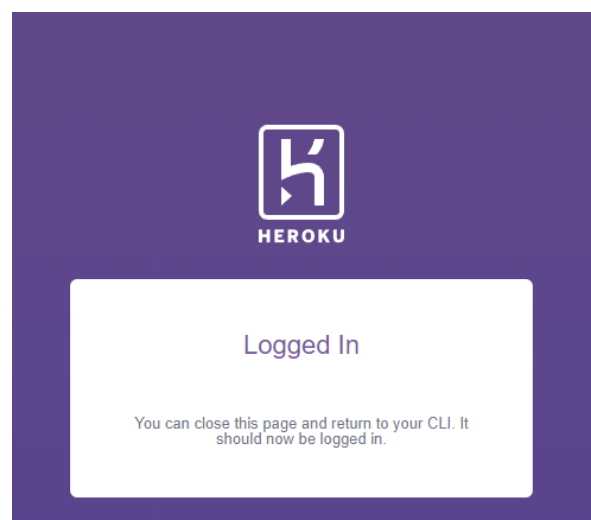
Para poder establecer contacto con Heroku se debe loguear con ayuda de su CLI, esto se logra ejecutando el siguiente comando en un `cmd`, `bash` o `terminal` y oprimiendo la tecla espacio:

```
heroku login
```

Esto abrirá una ventana en el navegador que permitirá loguearse:



Una vez *logueado*, 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.

### Inicializar Repositorio

Dado que *Heroku* maneja el código de despliegue utilizando un repositorio de *git*, se debe tener uno en el componente. Al momento de crear el proyecto con ayuda de *Vue CLI*, este inicializa un repositorio de *git* para el proyecto, esto resulta útil ya que evita la tarea de inicializar el repositorio.

## Agregar Remoto al Repositorio

El código fuente del componente será subido a un repositorio remoto administrado por Heroku, desde el cual se realizará el despliegue, 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 component"
```

## Realizar Despliegue

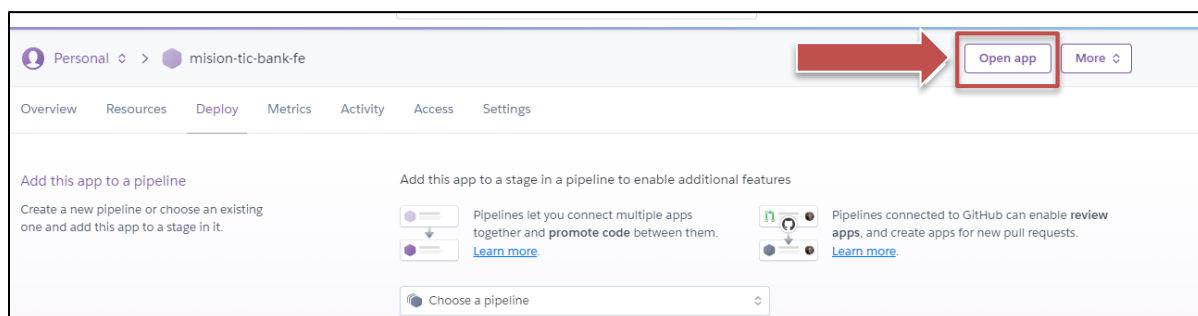
En este punto se puede completar el despliegue del componente, para esto únicamente bastará con subir el último commit al repositorio remoto de heroku. Esto se logra con el siguiente comando:

```
git push heroku master
```

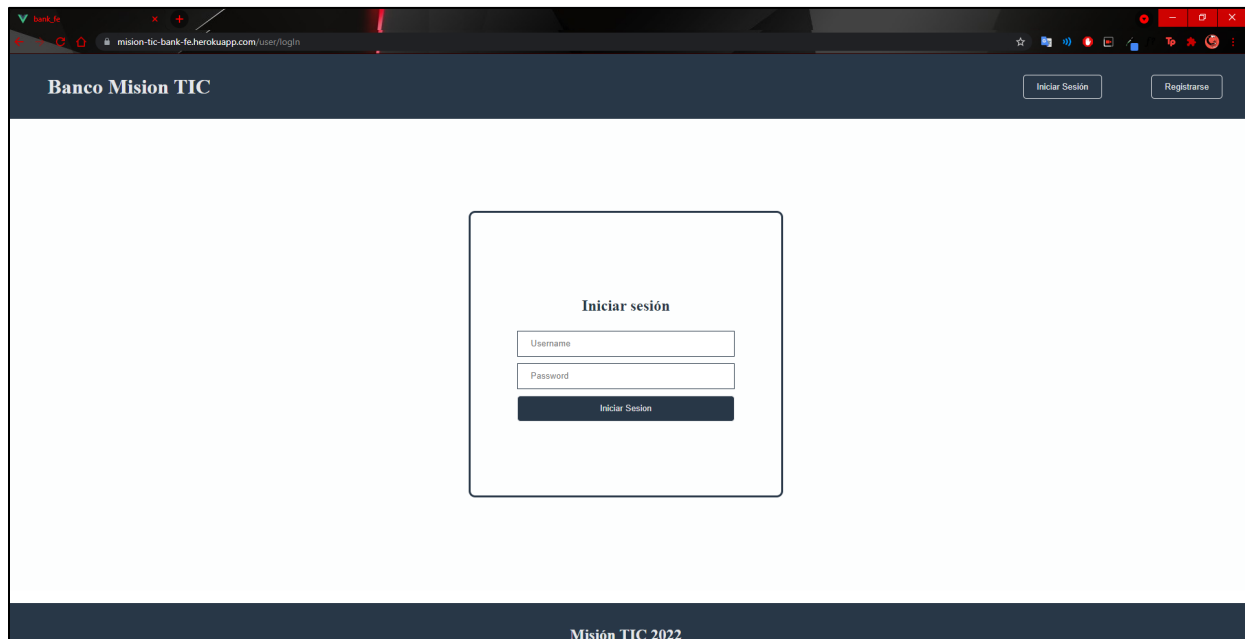
Al ejecutar este comando, se mostrará el estado del despliegue hasta finalizar.

## Acceder a la Aplicación

Una vez se ha realizado el *despliegue* de la aplicación se puede acceder a esta a través de su respectiva *URL*. Para ello, se debe dar click en el botón *Open app*, en el *dashboard* de Heroku:



Esto abrirá la página principal del componente de presentación desarrollado:



### Capa de Presentación: Pruebas

En la anterior sección, el componente de front-end fue *desplegado* de manera satisfactoria. Ahora se comprobará que existe una *comunicación* directa *entre* las diferentes *capas* de la aplicación, es decir, que la capa de presentación se comunica con la capa lógica y a la vez esta se comunica con la capa de datos. Para evidenciar esta interacción, se realizarán acciones en el componente front-end y se evidenciarán sus efectos en la base de datos y de manera inversa.

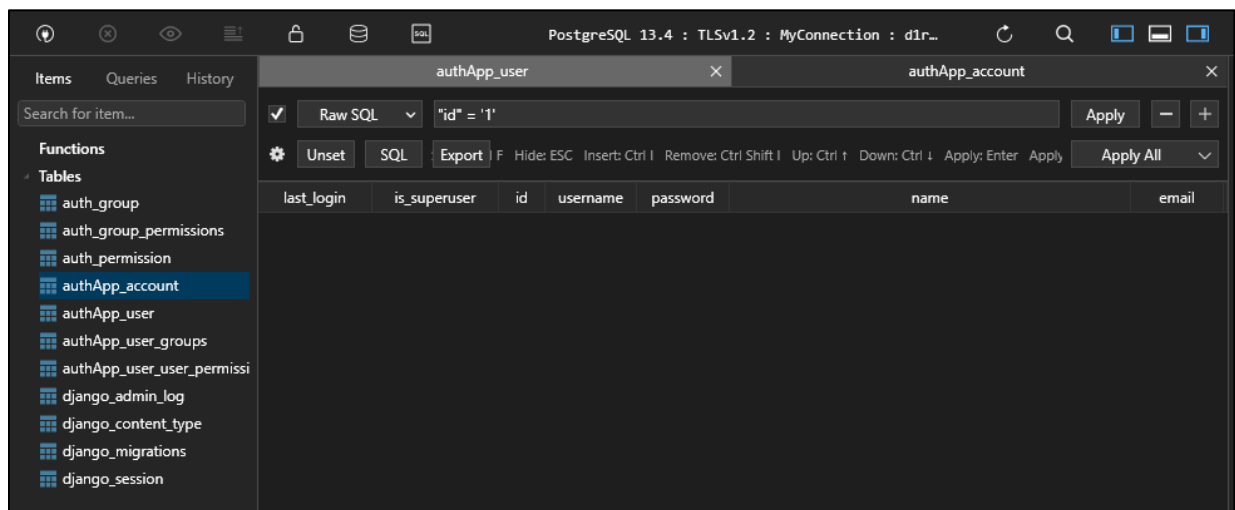
### Test #1: Creación de Usuario

En la primera prueba se busca evidenciar que la capa de presentación es capaz de comunicarse con las capas inferiores. Para esto se realizará un ejercicio, en el cual, en primer lugar, se revisará el estado actual de la base de datos con ayuda del cliente *TablePlus* (previamente instalado), para luego crear un usuario desde el componente de *frontend* y revisar nuevamente el estado de la *base de datos* para evidenciar los cambios.

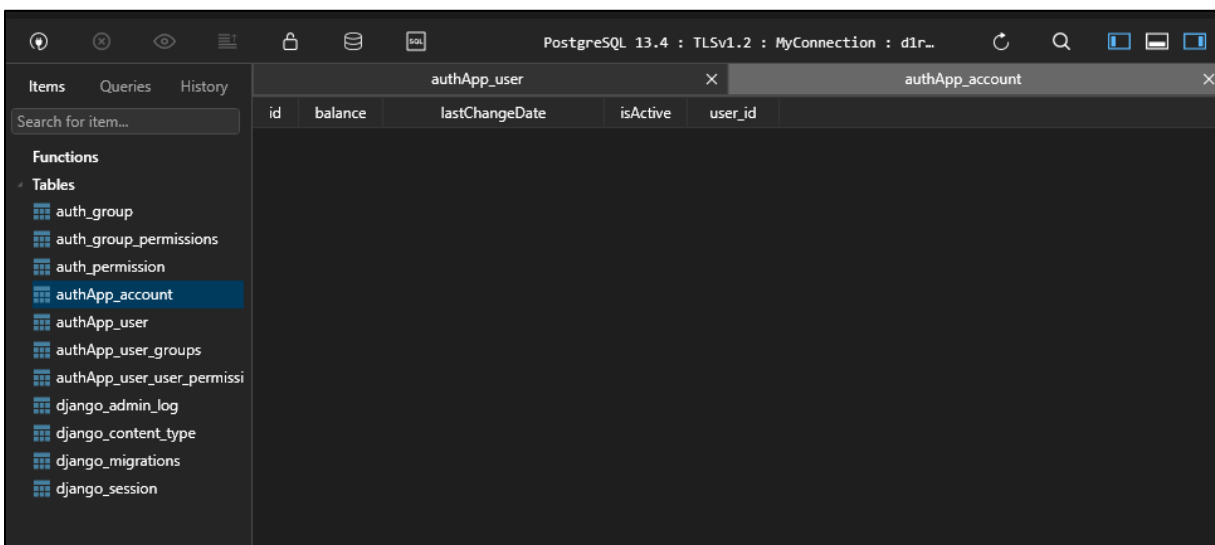
### Estado Inicial de la Base de Datos:

La tabla *User* se encuentra totalmente vacía:



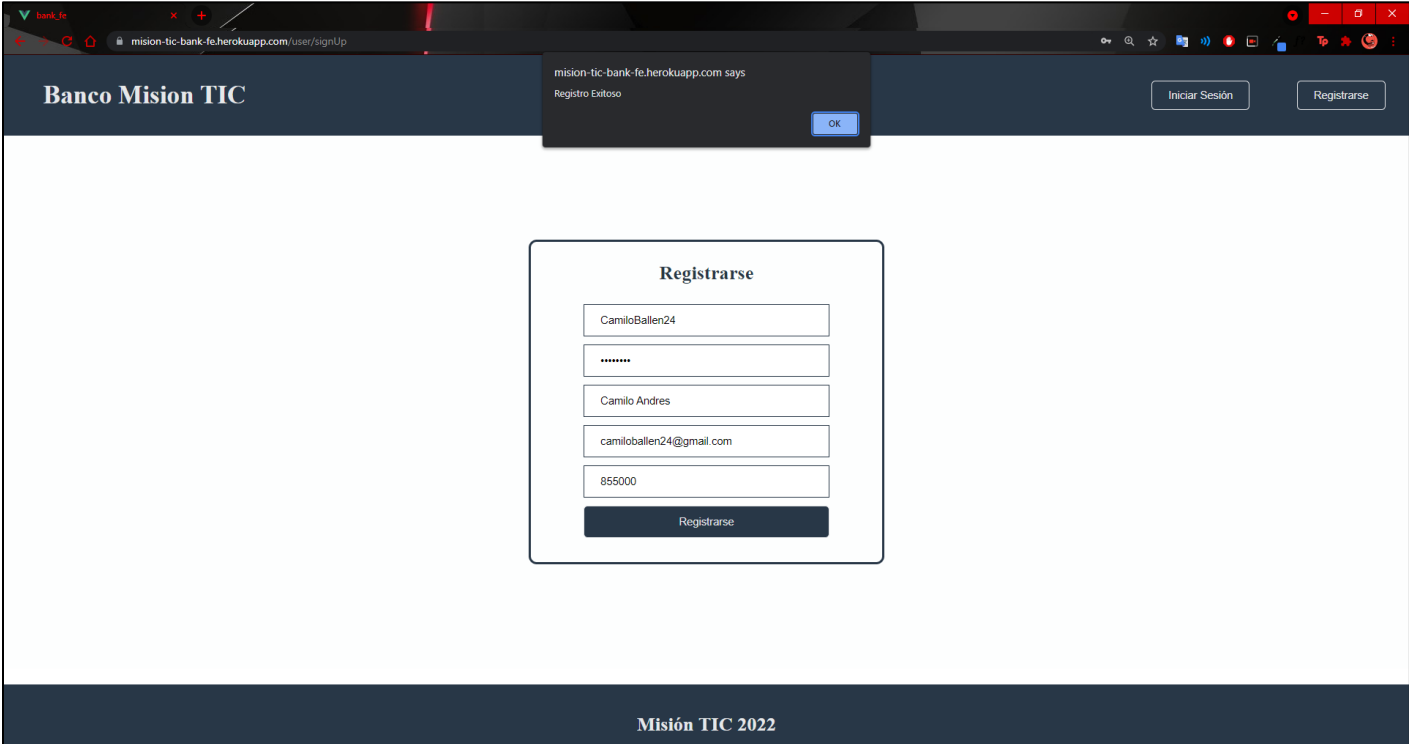


La tabla *Account*, también se encuentra totalmente vacía:



## Creación del Usuario:

Usando el componente *bank\_fe* recién desplegado se crea un usuario:



Banco Mision TIC

mision-tic-bank-fe.herokuapp.com says  
Registro Exitoso

OK

Registrar

Registrar

Misión TIC 2022

## Estado Posterior de la Base de Datos:

La tabla *User* tiene una nueva fila correspondiente al usuario recién creado:

PostgreSQL 13.4 : TLSv1.2 : MyConnection : d1r...

Items Queries History

Search for item...

Functions

Tables

- auth\_group
- auth\_group\_permissions
- auth\_permission
- authApp\_account
- authApp\_user
- authApp\_user\_groups
- authApp\_user\_user\_permissi
- django\_admin\_log
- django\_content\_type
- django\_migrations
- django\_session

Raw SQL

"id" = '1'

Apply

Unset SQL Export

Hide: ESC Insert: Ctrl I Remove: Ctrl Shift I Up: Ctrl ↑ Down: Ctrl ↓ Apply: Enter Apply

Apply All

last_login	is_superuser	id	username	password	name	email
NULL	FALSE	35	CamiloBal...	pbkdf2_sh...	Camilo Andres	camiloball...

Search for column...

last\_login timestampz

NULL

is\_superuser bool

f

id int8

35

username varchar

CamiloBallen24

password varchar

pbkdf2\_sha256\$260000

\$mMUj0DriK6vgtdIYepkxN

\$4HjJ882

+2bwcayO20kDehr1M9s7sIRkFJ

oq1WnwNV4=

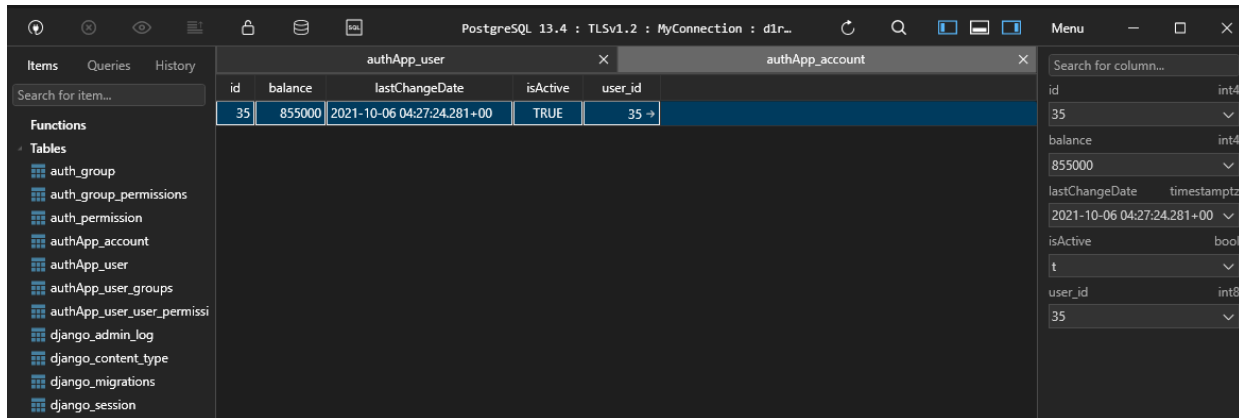
name varchar

Camilo Andres

email varchar

camiloballen24@gmail.com

La tabla Account tiene una nueva fila correspondiente al usuario recién creado:



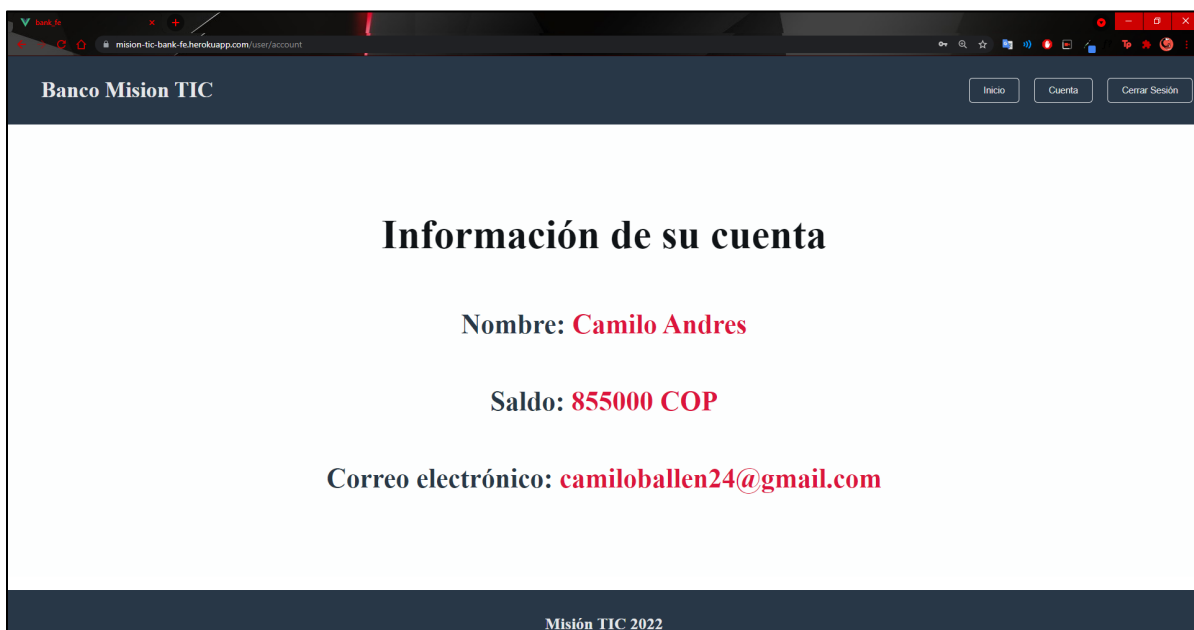
id	balance	lastChangeDate	isActive	user_id
35	855000	2021-10-06 04:27:24.281+00	TRUE	35

## Test #2: Creación de Usuario

En la segunda prueba se buscará evidenciar cómo la *capa de presentación* es capaz de actualizar la información si ocurre algún cambio en la información procesada y almacenada en las *capas inferiores*. Para esto se realizará un cambio desde la base de datos en la cuenta del usuario recién creado y luego se evidenciarán dichos cambios en el componente frontend.

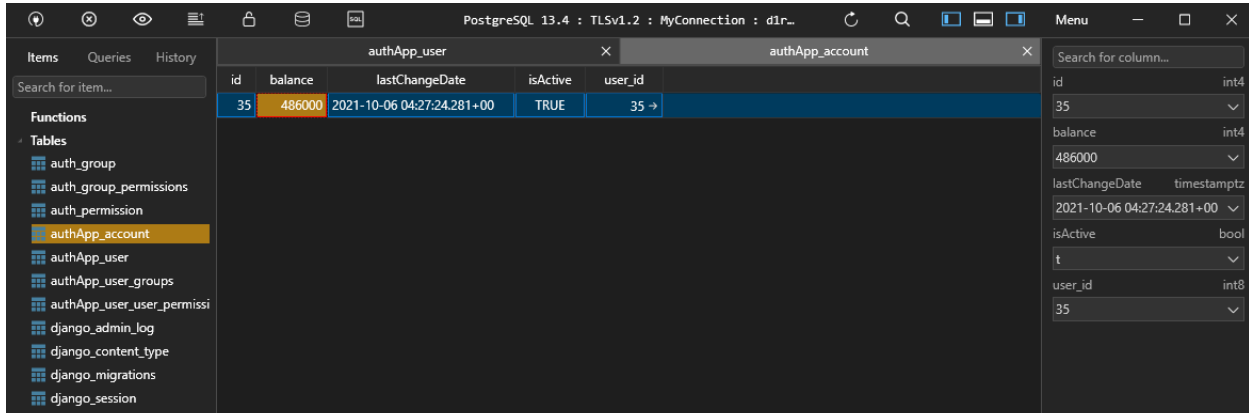
### Estado Actual del Componente Frontend:

Como se puede evidenciar el saldo actual del usuario es de *855.000 COP*.



## Modificar Valor en la Base de Datos:

En la tabla *Account* realizando doble click sobre la celda de *balance* (saldo) y editándola se puede modificar el saldo del usuario:



id	balance	lastChangeDate	isActive	user_id
35	486000	2021-10-06 04:27:24.281+00	TRUE	35 →

Para guardar los cambios, estos se deben subir haciendo click en el siguiente botón:



## Estado Final del Componente Front-End:

Como se puede evidenciar el saldo del usuario ha cambiado a *486.000 COP*



**Nota:** de ser necesario, en el material de la clase se encuentra una archivo *C3.AP.17. bank\_fe.rar*, con todos los avances en el desarrollo del componente realizado en esta guía.