



El futuro digital  
es de todos

MinTIC

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

```
( 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  
TIC 2022

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\_ibog@unateduco

# Componente de Presentación

## (Desarrollo Inicial)

### Actividad Práctica

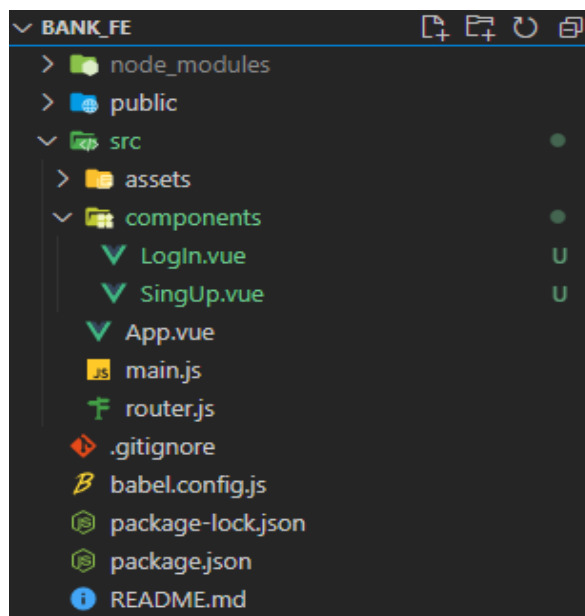
En esta guía se dará comienzo a la creación del *componente* de *front-end* llamado *bank\_fe*, este será desarrollado usando el *framework* *Vue*. Como punto de partida se tomará el proyecto base creado en la guía anterior.

El objetivo de esta guía es implementar dos funcionalidades básicas, la primera funcionalidad es permitir a los usuarios *iniciar sesión*, para esto deberán ingresar sus credenciales, en caso de que los usuarios no posean una cuenta, podrán hacer uso de la segunda funcionalidad, la cual permite a los usuarios *crear* una *cuenta* solamente con proveer algunos datos básicos.

Para llevar a cabo la implementación de las dos funcionalidades se deberá hacer uso de varios conceptos comunes en el desarrollo de componentes de front-end. Algunos de estos conceptos son el manejo de componentes de *Vue*, el trabajo con *routers* y el consumo de los servicios del componente de *back-end*.

### Estructura de Archivos

Para llevar a cabo el desarrollo de las funcionalidades de esta guía, es necesario crear dos nuevos archivos en el proyecto, estos archivos son *Login.vue* y *SignUp.vue* y se ubican en *src/components*, la estructura del proyecto debe lucir así:



Estos dos nuevos archivos corresponden a los componentes en los que se implementarán las dos funcionalidades, por lo cual cada componente tendrá una funcionalidad asociada.

## Router

El componente de front-end que se está desarrollando es un **componente WEB**, el cual será utilizado a través de un navegador, por lo cual se debe tener en cuenta una de las principales características del navegador que es la navegación a través de la **URL**, es decir acceder a recursos o cambiar el estado de la aplicación a través de interacciones con la URL, ya sea ingresando o modificando la URL del sitio o componente WEB.

**Vue** ofrece el concepto de **Router**, el cual tiene como objetivo manejar las URL dentro la aplicación. Para esto, parte de la idea básica de asociar una **URL** a un **Componente** de **Vue**, es decir, cada una de las funcionalidades implementadas en un componente tendrá una URL asociada. Esto genera la interacción deseada, ya que, al cambiar de URL la aplicación cambiará de componente y por consiguiente se accederá a otra funcionalidad.

Un punto a tener en cuenta es que la propia aplicación puede manipular la URL, y no solamente el usuario, es decir que la aplicación también puede cambiar de componentes y funcionalidades con ayuda del Router. La implementación de un Router es bastante sencilla ya que únicamente se deben asociar los componentes con sus respectivas URLs y agregar un nombre clave que pueda ser usado dentro de la aplicación.

El código correspondiente al Router que debe ir en el archivo **src/router.js**, es el siguiente:

```
import { createRouter, createWebHistory } from "vue-router";
import App from './App.vue';

import LogIn from './components/LogIn.vue'
import SignUp from './components/SignUp.vue'

const routes = [{
  path: '/',
  name: 'root',
  component: App
},
{
  path: '/user/logIn',
  name: "logIn",
  component: LogIn
},
{
  path: '/user/signUp',
  name: "signUp",
  component: SignUp
}
```

```
}  
];  
  
const router = createRouter({  
  history: createWebHistory(),  
  routes,  
});  
  
export default router;
```

Como se puede evidenciar, se establecieron 3 URLs:

- `/`: La cual corresponde al componente principal [APP](#).
- `/user/LogIn`: La cual corresponde al componente [LogIn](#).
- `/user/signUp`: La cual corresponde al componente [SignUp](#).

### Componente APP

El componente principal de cualquier aplicación de [Vue](#) es el componente [APP](#), ya que a partir de este se construye toda la aplicación, agregando de manera directa o indirecta los demás componentes. A continuación, se explicarán algunos conceptos necesarios para el desarrollo de este componente:

- **Router-View:** a través del elemento Router, Vue ofrece algunas herramientas que facilitan el desarrollo de la aplicación, en cuanto al manejo de la URL se refiere, la principal herramienta que ofrece Vue es [Router-View](#), esta herramienta es una etiqueta que se utiliza en el template y permite cargar de manera dinámica el componente correspondiente a la URL actual de la aplicación.

Normalmente cuando se quiere usar un componente dentro de otro se usa la etiqueta asociada al componente, por ejemplo, si se quiere usar el componente [LogIn](#) se debe usar una etiqueta de la forma `<LogIn></LogIn>`, pero esto no genera ninguna interacción con la URL, es decir, solamente carga el componente [LogIn](#) y nada más, a esto se le conoce como una carga estática. Por otro lado, la etiqueta `<router-view></router-view>` siempre cargará el componente asociado a la URL actual, y cuando se modifique la URL cambiará al componente cargado, a esto se le conoce como una carga dinámica.

- **Router-Push:** esta es otra de las herramientas que ofrece Vue para el manejo de las URL, [Router Push](#) permite cambiar la URL de la aplicación desde el script de un componente, esto resulta útil cuando se desea cambiar de componentes debido a algún evento ocurrido dentro de la aplicación.
- **Interacción entre Template y Script:** una de las principales ventajas de trabajar con un framework como Vue es la facilidad con la que se puede dotar de interacción a una simple estructura HTML, es

decir, que en lugar de ser una página totalmente estática, sea una página dinámica que se construya y modifique a partir de ciertos eventos y valores. Para lograr esto Vue ofrece varias herramientas, sin embargo, en esta guía solo se utilizarán dos de las más básicas:

- **`v-if="value"`:** este es un atributo que puede asociar a cualquier etiqueta del template, dicha etiqueta se cargará **únicamente** si el valor asignado es **True**. Este valor también se puede asignar por medio de una variable definida en la data del script del componente.
- **`v-on:click="function"`:** de manera similar a los eventos en JavaScript, en Vue se puede asociar una función al evento de un elemento. Sin embargo, en Vue es más sencillo, pues simplemente con añadir este atributo a una etiqueta, se asociará una función definida en el script, al evento **click** de la etiqueta.
- **Interacción entre Padre e Hijos:** Vue permite definir una estructura de árbol jerárquica entre los componentes, de tal manera que los componentes de un nivel superior (llamados padres), pueden controlar a los componentes de un nivel inferior (llamados hijos), esto permite estructurar y ordenar de manera fácil los componentes de una aplicación. Como es de esperarse existe una comunicación entre padres e hijos, para esto Vue ofrece dos conceptos principales:
  - **Props:** son un mecanismo en el cual un componente padre le puede pasar un valor a un componente hijo al momento de definirlo. Esto permite que el hijo realice operaciones a partir de cierta información provista por su padre.
  - **Emit:** Uno de los mecanismos más interesantes de Vue es ejecutar funciones del componente padre desde los componentes hijos, es decir que, un componente padre puede definir una función, pero puede ser un componente hijo quien la ejecute o emita. Esto resulta útil, pues si bien el hijo puede controlar el momento de ejecución, el padre tiene el control sobre lo que se ejecuta, ya que en este se realiza la implementación de la función. Cuando se define el componente hijo, el componente padre debe indicar qué funciones puede ejecutar su hijo.

El componente **APP** es el encargado de gestionar el flujo de la aplicación y sus componentes, por ello su desarrollo se realizará por partes. En la guía anterior se definió una estructura inicial, que a continuación se editará y que en la próxima guía se complementará.

El código del componente **APP** que debe ir en el archivo **src/App.vue** es el siguiente:

## Template del Componente APP:

```
<template>
  <div id="app" class="app">

    <div class="header">

      <h1>Banco UN</h1>
      <nav>
        <button v-if="is_auth" > Inicio </button>
        <button v-if="is_auth" > Cuenta </button>
        <button v-if="is_auth" > Cerrar Sesión </button>
        <button v-if="!is_auth" v-on:click="loadLogIn" > Iniciar Sesión </button>
        <button v-if="!is_auth" v-on:click="loadSignUp" > Registrarse </button>
      </nav>
    </div>

    <div class="main-component">
      <router-view
        v-on:completedLogIn="completedLogIn"
        v-on:completedSignUp="completedSignUp"
      >
    </router-view>
    </div>

    <div class="footer">
      <h2>Misión TIC 2022</h2>
    </div>

  </div>
</template>
```

Algunos detalles para tener en cuenta del *template* del componente APP son los siguientes:

- El componente APP servirá de base de la aplicación y sobre este se cargarán todos los demás componentes dependiendo de la URL. Por ello se hará uso de un *router view* para cargar los componentes hijos. Además, en el componente APP se definen algunas etiquetas que estarán en todo momento durante la ejecución de la aplicación: El *header* y el *footer*.
- El *nav* del componente está compuesto por 5 botones que representan algunas funcionalidades de la aplicación, pero como se puede evidenciar estos botones poseen el atributo *v-if*, por lo cual, se cargarán de manera condicional, dependiendo del valor de la variable *is\_auth*.

- Los botones *Iniciar Sesión* y *Registrarse* tienen asociada una función al evento *click*, esta función deberá modificar la ruta para cargar el componente correspondiente en el *router view*.
- El componente APP delegará a componentes hijos la implementación de las funcionalidades *Login* y *SignUp*, pero este quiere controlar el resultado de estas operaciones, para ello definirá dos métodos *completedLogin* y *completedSignUp*, los cuales sus hijos podrán emitir para informar que se ha finalizado el proceso. Dado que sus componentes hijos se cargarán en el *router view*, es en este en el que se indicarán los métodos que sus hijos pueden emitir.

### Script del Componente APP:

```
<script>
export default {
  name: 'App',

  data: function(){
    return{
      is_auth: false
    }
  },

  components: {
  },

  methods:{
    verifyAuth: function() {
      if(this.is_auth == false)
        this.$router.push({name: "logIn"})
    },

    loadLogIn: function(){
      this.$router.push({name: "logIn"})
    },

    loadSignUp: function(){
      this.$router.push({name: "signUp"})
    },

    completedLogIn: function(data) {},

    completedSignUp: function(data) {},

  },

  created: function(){
```



```
this.verifyAuth()  
}  
  
}  
</script>
```

Algunos detalles para tener en cuenta del *script* del componente APP son:

- En la data se realiza la definición de *is\_auth* con el valor falso por defecto. Esta variable ayudará a determinar si un usuario ya se ha autenticado o no, lo cual será útil para próximas funcionalidades.
- Se define la función *created*, la cual se ejecuta cada vez que se renderiza el componente. En esta se realiza un llamado a la función *verifyAuth*, la cual tiene como objetivo determinar si un usuario está autenticado o no y, en caso de no estarlo, cargará el componente *LogIn* para que el usuario realice el proceso de autenticación. En este punto del desarrollo la función *verifyAuth* está implementada de manera parcial, conforme avance el desarrollo del componente se complementará su código.
- Las funciones *loadLogIn* y *loadSignUp*, tendrán como objetivo cambiar la URL de la aplicación para cargar el componente correspondiente, para ello harán uso de *Router Push*.
- Se definen las funciones *completedLogIn* y *completedSignUp*, pero su implementación se realizará en la próxima guía. Estas funciones serán llamadas por los componentes hijos cuando completen sus funcionalidades.

### Style del Componente APP:

```
<style>  
  
body{  
  margin: 0 0 0 0;  
}  
  
.header{  
  margin: 0%;  
  padding: 0;  
  width: 100%;  
  height: 10vh;  
  min-height: 100px;  
  
  background-color: #283747 ;  
  color:#E5E7E9 ;  
  
  display: flex;
```





```
justify-content: space-between;
align-items: center;
}

.header h1{
width: 20%;
text-align: center;
}

.header nav {
height: 100%;
width: 20%;

display: flex;
justify-content: space-around;
align-items: center;

font-size: 20px;
}

.header nav button{
color: #E5E7E9;
background: #283747;
border: 1px solid #E5E7E9;

border-radius: 5px;
padding: 10px 20px;
}

.header nav button:hover{
color: #283747;
background: #E5E7E9;
border: 1px solid #E5E7E9;
}

.main-component{
height: 75vh;
margin: 0%;
padding: 0%;

background: #FDFEFE ;
}

.footer{
margin: 0;
padding: 0;
```

```
width: 100%;
height: 10vh;
min-height: 100px;

background-color: #283747;
color: #E5E7E9;

}

.footer h2{
width: 100%;
height: 100%;

display: flex;
justify-content: center;
align-items: center;
}
</style>
```

## Componente Login

Una vez se ha creado la base de la aplicación con ayuda del componente APP, se pueden implementar las funcionalidades. La primera funcionalidad a implementar es [Login](#), esta permitirá a los usuarios autenticarse. Para implementar esto es necesario tener claro algunos conceptos:

- **Servicios del Back-End:** la capa de presentación es la encargada de interactuar directamente con el usuario final, para hacer esto posible debe comunicarse con la capa de lógica y consumir los servicios que esta ofrece. Para implementar la funcionalidad de [Login](#) la capa de lógica ofrece un [endpoint](#) que el componente [bank\\_fe](#) deberá utilizar a través de una petición [HTTP](#), es decir, el componente de front-end deberá asumir el rol de [cliente](#) y realizar una petición al componente de backend que asumirá el rol de [servidor](#).

Para consumir los servicios del componente de back-end en Vue se hace uso de la herramienta [Axios](#), esta permite realizar y manejar peticiones [HTTP](#) de una manera rápida y sencilla. Para hacer uso de esta herramienta en el proyecto, simplemente se debe instalar, lo cual se logra ejecutando el siguiente comando en una consola ubicada en la raíz del proyecto:

```
npm install axios
```

- **Formularios:** el principal objetivo de la capa de presentación es recolectar información del usuario de una manera rápida y sencilla para ser procesada por las demás capas. Un elemento muy común en el desarrollo de componentes de frontend son los formularios, estos son estructuras que permiten

recolectar y manejar información de manera sencilla. Vue entiende la importancia de estos y brinda dos herramientas que facilitan el trabajo con formularios:

- **v-model**: Vue permite asociar las entradas de un formulario ubicado en el **template** de un componente, con un objeto ubicado en el **script**. Para hacer esto posible simplemente se debe definir el objeto en la data del script y con ayuda del atributo **v-model** asociarlo con la entrada correspondiente, Vue se encarga de mantener actualizado los valores en ambos lugares.
- **Submit**: los formularios suelen tener asociado un botón que permite enviar el formulario para ser procesado, Vue ofrece un atributo que permite asociar una función a dicho evento, con ayuda del atributo **v-on:submit.prevent="functionName"** en el formulario, se le indica a Vue que se ejecute una determinada función cuando el usuario realice el envío del formulario.

El código del componente **Login** que debe ir en el archivo **src/components/Login.vue** es el siguiente:

### Template del Componente Login:

```
<template>

  <div class="logIn_user">
    <div class="container_logIn_user">
      <h2>Iniciar sesión</h2>

      <form v-on:submit.prevent="processLogInUser" >
        <input type="text" v-model="user.username" placeholder="Username">
        <br>
        <input type="password" v-model="user.password" placeholder="Password">
        <br>
        <button type="submit">Iniciar Sesion</button>
      </form>
    </div>
  </div>

</template>
```

El **template** es relativamente simple, es un formulario asociado a una función llamada **processLogInUser** que detectará cuando el usuario lo envíe. Dicho formulario tiene dos entradas que son **username** y **password**, estas están asociadas al objeto **user**.

## Script del Componente Login:

```
<script>
import axios from 'axios';

export default {
  name: "LogIn",

  data: function(){
    return {
      user: {
        username:"",
        password:""
      }
    },
  },

  methods: {
    processLogInUser: function(){
      axios.post(
        "https://mision-tic-bank-be.herokuapp.com/login/",
        this.user,
        {headers: {}}
      )
      .then((result) => {
        let dataLogIn = {
          username: this.user.username,
          token_access: result.data.access,
          token_refresh: result.data.refresh,
        }

        this.$emit('completedLogIn', dataLogIn)
      })
      .catch((error) => {

        if (error.response.status == "401")
          alert("ERROR 401: Credenciales Incorrectas.");

      });
    }
  }
}
</script>
```

Algunos detalles para tener en cuenta del *script* del componente *LogIn* son:

- Se debe notar que la primera línea del *Script* se encarga de importar la herramienta *axios* para su posterior uso.
- En la definición de la data se define el objeto *user*, este contendrá los datos de entrada del formulario.
- El propósito del componente *Login* se lleva a cabo en la función *processLoginUser*. En esta se puede evidenciar cómo se hace uso de la herramienta *axios* para ejecutar una petición *HTTP* de tipo *POST* al componente de *back-end* usando la *URL* correspondiente, pasando como parámetro las credenciales del usuario. Luego, se procesa la respuesta del *back-end*, si todo es correcto se crea un objeto que servirá de parámetro para emitir la función del padre *completedLogin*, esta le indicará al componente APP que se ha realizado de manera correcta la autenticación; por otro lado, si falla la petición, se capturan y procesan los errores.

### Style del Componente Login:

```
<style>

.logIn_user{
  margin: 0;
  padding: 0%;
  height: 100%;
  width: 100%;

  display: flex;
  justify-content: center;
  align-items: center;
}

.container_logIn_user {
  border: 3px solid #283747;
  border-radius: 10px;
  width: 25%;
  height: 60%;

  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

.logIn_user h2{
  color: #283747;
}

}
```

```
.logIn_user form{
  width: 70%;
}

.logIn_user input{
  height: 40px;
  width: 100%;

  box-sizing: border-box;
  padding: 10px 20px;
  margin: 5px 0;

  border: 1px solid #283747;
}

.logIn_user button{
  width: 100%;
  height: 40px;

  color: #E5E7E9;
  background: #283747;
  border: 1px solid #E5E7E9;

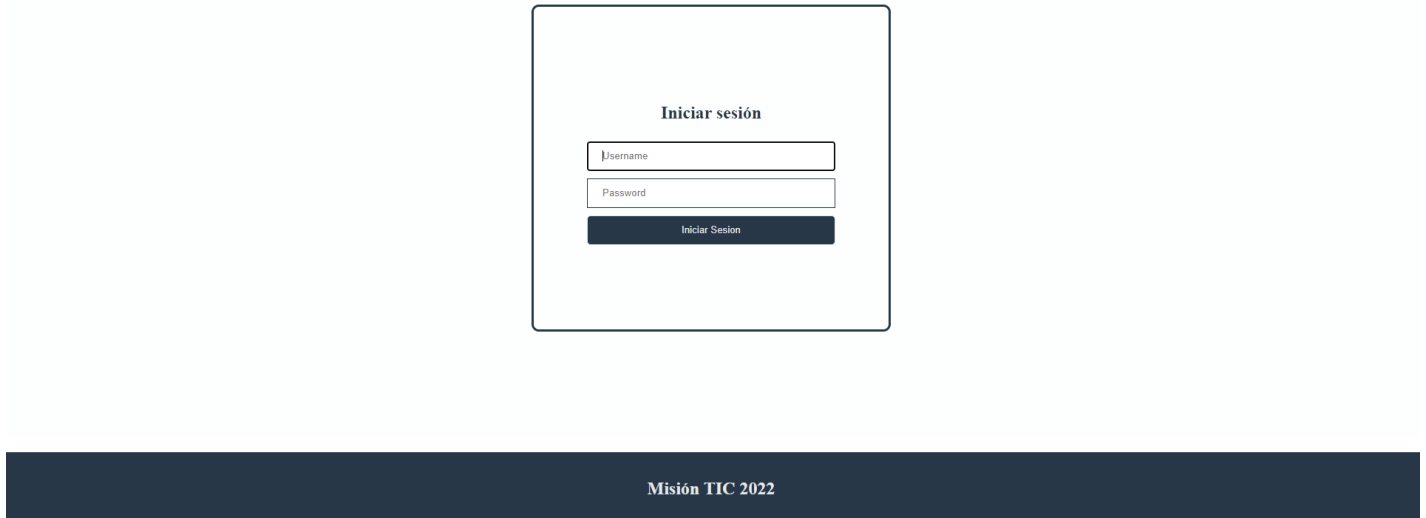
  border-radius: 5px;
  padding: 10px 25px;
  margin: 5px 0;
}

.logIn_user button:hover{
  color: #E5E7E9;
  background: crimson;
  border: 1px solid #283747;
}

</style>
```

### Resultado del Componente Login:

Ahora, si se ejecuta el servidor del componente de presentación y se dirige a la URL <http://localhost:8080/user/login> , se encontrará lo siguiente:



## Componente SignUp

Por último, se llevará a cabo la implementación de la funcionalidad de registro usando el componente [SignUp](#), esta funcionalidad permitirá a los usuarios registrarse con solo proveer algunos datos. Su estructura es muy similar a la del componente [Login](#), se utilizan los mismos conceptos. Se debe recordar que este componente se carga cuando se hace [clic](#) sobre el botón de registrarse.

El código del componente [SignUp](#) que debe ir en el archivo [SignUp.vue](#) es el siguiente:

### Template del Componente SignUp:

```
<template>

  <div class="signUp_user">
    <div class="container_signUp_user">
      <h2>Registrarse</h2>

      <form v-on:submit.prevent="processSignUp" >
        <input type="text" v-model="user.username" placeholder="Username">
        <br>
      </form>
    </div>
  </div>
</template>
```



```

    <input type="password" v-model="user.password" placeholder="Password">
    <br>

    <input type="text" v-model="user.name" placeholder="Name">
    <br>

    <input type="email" v-model="user.email" placeholder="Email">
    <br>

    <input type="number" v-
model="user.account.balance" placeholder="Initial Balance">
    <br>

    <button type="submit">Registrarse</button>
  </form>
</div>

</div>
</template>

```

### Script del Componente SignUp:

```

<script>
import axios from 'axios';

export default {
  name: "SignUp",

  data: function(){
    return {
      user: {
        username: "",
        password: "",
        name: "",
        email: "",
        account: {
          lastChangeDate: (new Date()).toISOString(),
          balance: 0,
          isActive: true
        }
      }
    }
  },

  methods: {

```

```
processSignUp: function(){
  axios.post(
    "https://mision-tic-bank-be.herokuapp.com/user/",
    this.user,
    {headers: {}}
  )
  .then((result) => {
    let dataSignUp = {
      username: this.user.username,
      token_access: result.data.access,
      token_refresh: result.data.refresh,
    }

    this.$emit('completedSignUp', dataSignUp)
  })
  .catch((error) => {
    console.log(error)
    alert("ERROR: Fallo en el registro.");
  });
}
}
</script>
```

La estructura es muy similar a la del componente [SignUp](#), salvo por algunos detalles como la manera en la se obtiene la fecha actual usando valores que ofrece JavaScript y dándole el formato adecuado.

### Style del Componente SignUp:

```
<style>

.signUp_user{
  margin: 0;
  padding: 0%;
  height: 100%;
  width: 100%;

  display: flex;
  justify-content: center;
  align-items: center;
}

.container_signUp_user {
```



```
border: 3px solid #283747;
border-radius: 10px;
width: 25%;
height: 60%;

display: flex;
flex-direction: column;
justify-content: center;
align-items: center;
}

.signUp_user h2{
  color: #283747;
}

.signUp_user form{
  width: 70%;
}

.signUp_user input{
  height: 40px;
  width: 100%;

  box-sizing: border-box;
  padding: 10px 20px;
  margin: 5px 0;

  border: 1px solid #283747;
}

.signUp_user button{
  width: 100%;
  height: 40px;

  color: #E5E7E9;
  background: #283747;
  border: 1px solid #E5E7E9;

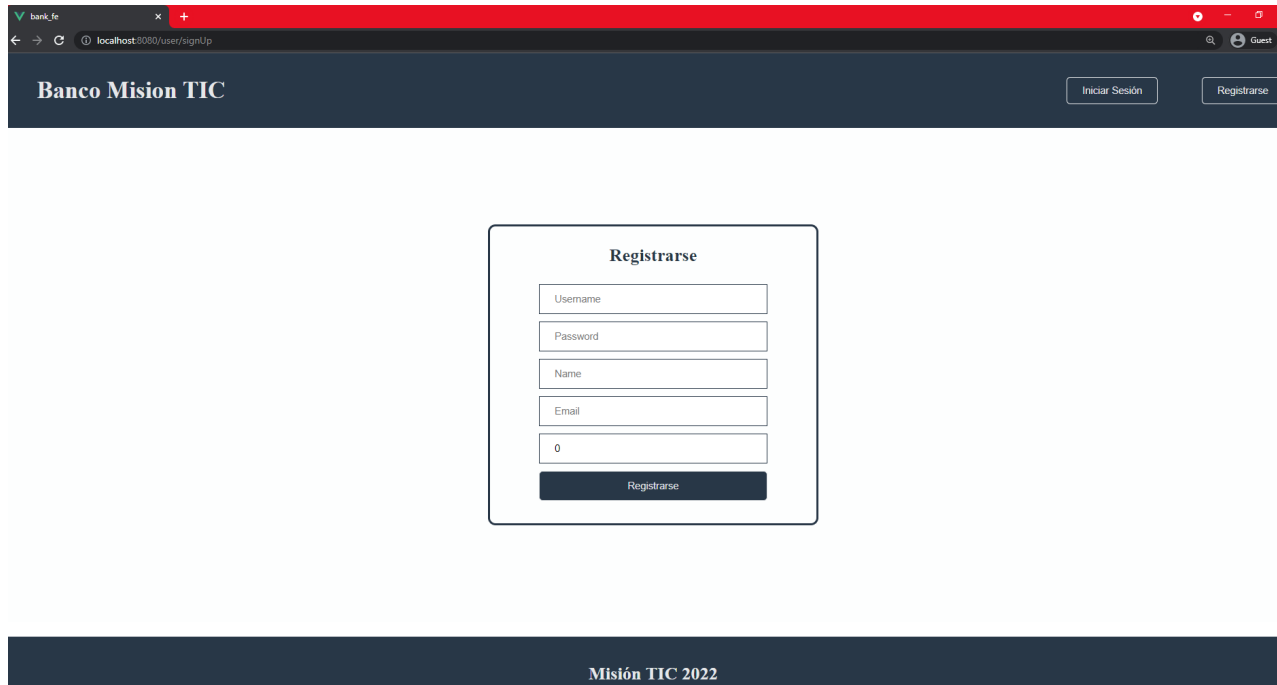
  border-radius: 5px;
  padding: 10px 25px;
  margin: 5px 0 25px 0;
}

.signUp_user button:hover{
  color: #E5E7E9;
  background: crimson;
```

```
border: 1px solid #283747;
}
</style>
```

## Resultado del Componente SignUp:

Ahora, si se ejecuta el servidor del componente de presentación y se dirige a la URL <http://localhost:8080/user/signUp> , se encontrará lo siguiente:



**Nota:** de ser necesario, en el material de la clase se encuentra un archivo *C3.AP.15. bank\_fe.rar*, con todos los avances en el desarrollo del componente realizado en esta guía.