El futuro digital es de todos

MinTIC

Ciclo 4a:

Desarrollo de aplicaciones web

UNIVERSIDAD NACIONAL DE COLOMBIA

MisiónTIC2022

VERSIÓN 1.0

Unidad de educación continua y permanente
Facultad de Ingeniería

Unidad Camilo Torres
Calle 44 # 45-67
Bloque B5 piso 1

(57) + 316 5000
uec_ibog@unaleduco

Actividad Práctica

(Componente Web - Parte 3)

En la guía anterior se realizó un proceso de evolución de software en el componente web, este proceso consistió en reemplazar la implementación de algunas funcionalidades; se pasó de consumir un componente que exponía una *API* de tipo *REST*, a consumir una *API Gateway* que expone sus servicios a través de una *API* de tipo *GraphQL*. El anterior proceso de actualizar la implementación de funcionalidades es uno de los casos más comunes de *evolución* de un *sistema* de *software*. En esta guía se trabajará en otro de los casos más comunes, el cual consiste en agregar nuevas funcionalidades al sistema de software.

A lo largo de la guía se agregarán dos módulos enfocados al manejo de la información bancaria de los usuarios. El primero será implementado en el archivo *Account.vue* y se encargará de mostrar toda la información bancaria del usuario, el segundo se implementará en el archivo *Transaction.vue* y permitirá realizar transacciones entre los usuarios.

Módulo Account

En el *ciclo 3* se implementaron ciertas funcionalidades en el componente *Account*, sin embargo, debido al cambio en la estructura del sistema de software, es necesario reestructurarlo completamente. Ahora, este componente dispondrá de dos funcionalidades para mostrar la información bancaria del usuario, la primera consistirá en obtener el saldo actual y el último movimiento realizado, y la segunda funcionalidad consistirá en mostrar el historial de todas las transacciones en las que el usuario ha estado involucrado.

El código correspondiente al *template* del archivo *Account.vue* es el siguiente:

```
<template>
  <div id="Historial">

    <div class="container">
      <h2>
        Titular Cuenta:
        <span>{{ username }}</span>
      </h2>
      <h2>
        Saldo:
        <span>${{ accountByUsername.balance }} COP</span>
      </h2>
      <h2>
        Último Movimiento:
        <span>
          {{ accountByUsername.lastChange.substring(0, 10) }}
```

```
          {{ accountByUsername.lastChange.substring(11, 19) }}
        </span>
      </h2>
    </div>

    <h2>Transacciones</h2>
    <div class="container-table">
        <table>
            <tr>
                <th>Fecha</th>
                <th>Hora</th>
                <th>Origen</th>
                <th>Destino</th>
                <th>Valor</th>
            </tr>

            <tr v-for="transaction in transactionByUsername" :key="transaction.id">
                <td>{{ transaction.date.substring(0, 10) }}</td>
                <td>{{ transaction.date.substring(11, 19) }}</td>
                <td>{{ transaction.usernameOrigin }}</td>
                <td>{{ transaction.usernameDestiny }}</td>
                <td>${{ transaction.value }} COP</td>
            </tr>
        </table>
    </div>
  </div>

</template>
```

El *template* es un poco complejo, pero realmente está compuesto por elementos sencillos:

- En primer lugar, aparece un *div* con la clase *container*, este se encarga de agrupar algunos campos que muestran información bancaria relevante para el usuario, en esta información se encuentra el nombre de usuario, el saldo y la fecha del último movimiento realizado por el usuario.
- En segundo lugar, se tiene un *div* con la clase *container-table*, que agrupa una *table* donde se mostrarán cada una de las transacciones en las que el usuario se ha visto involucrado. Debido a que no se conoce el número de transacciones, se hace uso de la instrucción *v-for*, esta creará una fila por cada uno de los elementos en el objeto *transactionByUsername*.

El código correspondiente al *script* del archivo *Account.vue* es el siguiente:

```
<script>
import gql from "graphql-tag";

export default {
```

```
name: "Account",

data: function () {
  return {
    username: localStorage.getItem("username") || "none",
    transactionByUsername: [],
    accountByUsername: {
      balance: "",
      lastChange: "",
    }
  };
},

apollo: {
  transactionByUsername: {
    query: gql`
      query ($username: String!) {
        transactionByUsername(username: $username) {
          id
          usernameOrigin
          usernameDestiny
          value
          date
        }
      }
    `,
    variables() {
      return {
        username: this.username,
      };
    },
  },

  accountByUsername: {
    query: gql`
      query ($username: String!) {
        accountByUsername(username: $username) {
          balance
          lastChange
        }
      }
    `,
    variables() {
      return {
        username: this.username,
      };
```

```
      },
    },
  },

  created: function () {
    this.$apollo.queries.transactionByUsername.refetch();
    this.$apollo.queries.accountByUsername.refetch();
  }
};
</script>
```

En este caso el *script* es bastante simple, dado que ya se ha trabajado con una estructura similar en la guía pasada. Se tienen dos *Queries* que *Apollo* ejecutará justo después de renderizar el componente, y cargará la información obtenida de estas operaciones en los objetos correspondientes, estos objetos luego son usados para mostrar la información en el *template*. Las *Queries* están ligadas a las funcionalidades implementadas, el primer elemento se llama *accountByUsername* y contiene la información básica de la cuenta bancaria del usuario, el segundo elemento es *transactionByUsername* y contiene todas las transacciones donde el usuario se ha visto involucrado.

Adicionalmente, se crea una función en *created* que se ejecutará cada vez que se renderice el componente en pantalla. En dicha función se hace un *refetch()* de ambas Queries, de forma que cada vez que se acceda al componente se actualice la información bancaria del usuario.

Finalmente, el código correspondiente al *style* del archivo *Account.vue* es el siguiente:

```
<style>
#Historial {
  width: 100%;

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

#Historial .container-table{
    width:50%;

    max-height: 250px;
    overflow-y: scroll;
    overflow-x: hidden;
}
```

```css
#Historial table {
  width: 100%;
  border-collapse: collapse;
  border: 1px solid rgba(0, 0, 0, 0.3);

}

#Historial table td,
#Historial table th {
  border: 1px solid #ddd;
  padding: 8px;
}

#Historial table tr:nth-child(even) {
  background-color: #f2f2f2;
}

#Historial table tr:hover {
  background-color: #ddd;
}

#Historial table th {
  padding-top: 12px;
  padding-bottom: 12px;
  text-align: left;
  background-color: crimson;
  color: white;
}

#Historial > h2 {
  color: #283747;
  font-size: 25px;
}

#Historial .container {
  padding: 30px;
  border: 3px solid rgba(0, 0, 0, 0.3);
  border-radius: 20px;
  margin: 5% 0 1% 0;
}

#Historial  .container h2 {
  font-size: 25px;
  color: #283747;
}
#Historial .container span {
```

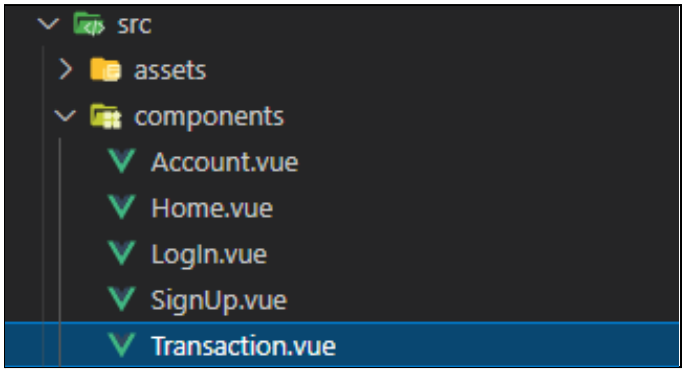
```
  color: crimson;
  font-weight: bold;
}
</style>
```

Una vez modificado el componente *Account.vue*, ya se puede ejecutar el servidor, y se puede observar el resultado de los cambios:

Módulo Transaction

Por último, se realizará la implementación de la funcionalidad que permite realizar transacciones entre los usuarios de la aplicación, la cual se podría considerar el núcleo del sistema. Para desarrollar esta funcionalidad se debe crear un nuevo componente desde cero e integrarlo a la aplicación.

Primero que todo se debe crear un archivo llamado *Transaction.vue* en la carpeta de componentes, en este archivo se desarrollarán las diferentes partes del componente:



Router

Ahora que se tiene el archivo en el que irá el componente *Transaction*, se debe crear una ruta para este. Para ello, se deben incluir dos fragmentos de código en el archivo *src/router.js*.

Primero se debe importar el componente esto se logra con la siguiente línea de código, debe ir a la altura de las demás importaciones:

```
import Transaction from './components/Transaction.vue'
```

Luego se debe crear la ruta para el componente, eso se logra con el siguiente fragmento de código, debe ir a la altura de las demás rutas:

```
  {
      path: '/user/transaction',
      name: "transaction",
      component: Transaction
  }
```

Componente Principal

Ahora que se tiene una ruta que permite la navegación, se debe definir un mecanismo que permita acceder al componente, para esto bastará con realizar algunos pequeños cambios en el componente principal (*App.vue*).

Primero se debe agregar un botón que permita acceder al componente, esto se logra agregando la siguiente línea en el *template*, el botón debe ir entre el botón *Cuenta* y el botón *Cerrar Sesión*. De esta forma, el código del *template* debe ser el siguiente:

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

    <div class="header">

      <h1> Banco Misión TIC </h1>
      <nav>
        <button v-if="is_auth" v-on:click="loadHome"> Inicio </button>
        <button v-if="is_auth" v-on:click="loadAccount"> Cuenta </button>
        <button v-if="is_auth" v-on:click="loadTransaction"> Transacción </button>
        <button v-if="is_auth" v-on:click="logOut"> 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"
        v-on:logOut="logOut"
      >
      </router-view>
    </div>


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

  </div>
</template>
```

Este botón, invoca una función que cargará la ruta en la que se encuentra el componente *Transaction*, esta función se debe implementar en la sección de métodos del *script*, y su código es el siguiente:

```
    loadTransaction: function(){
      this.$router.push({ name: "transaction" });
    }
```

Por último, dado que se agregó un nuevo botón se debe incrementar el tamaño del *nav bar*, esto se logra con realizando el siguiente cambio en el *style*:

```
.header nav {
  height: 100%;
  width: 30%;

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

  font-size: 20px;
}
```

Algo a tener en cuenta es que muchos de los fragmentos de código corresponden a elementos de arreglos, para evitar conflictos es importante controlar el uso correcto de las comas.

Componente Transaction

Ahora que se tienen todos los elementos necesarios, se puede trabajar directamente en el componente *Transaction*. Para permitir realizar transacciones entre los usuarios se utilizará un mecanismo similar al trabajado en el *registro* y *autenticación* en la guía anterior, es decir, se tendrá un formulario enlazado a un objeto que almacenará los datos ingresados en los campos del formulario, y al realizar el envío del formulario se invocará una función, la cual tomará dicho objeto y procesará el envío del formulario. En dicho procesamiento se realizará la actualización del *access token* utilizando el *refresh token*, y posteriormente se realizará la transacción, de esta forma se asegura que la transacción no sea rechazada por tener un *access token* caducado.

El código correspondiente al *template* del componente *Transaction.vue* es el siguiente:

```
<template>

  <div id="Transaction" class="transaction">
    <div class="container_transaction">
      <h2>Realizar Transacción</h2>

      <form v-on:submit.prevent="processTransaction">
        <input
          type="text"
          v-model="createTransaction.usernameDestiny"
          placeholder="Usuario Destino"
        />
```

```
        <br />
        <input
          type="number"
          v-model="createTransaction.value"
          placeholder="Valor"
        />
        <br />
        <button type="submit">Realizar Transacción</button>
      </form>
    </div>
  </div>

</template>
```

El *template* es sencillo, un formulario con dos campos, el primero es el *nombre de usuario* del usuario al que se le realizará la *transacción* y el segundo es el *monto* de la *transacción*, ambos campos están conectados con un objeto usando un *v-model*.

El código correspondiente al *script* del componente *Transaction.vue* es el siguiente:

```
<script>
import gql from "graphql-tag";

export default {
  name: "Transaction",

  data: function() {
    return {
      createTransaction: {
        usernameOrigin: localStorage.getItem("username"),
        usernameDestiny: "",
        value: "",
      },
    };
  },

  methods: {
    processTransaction: async function() {

      if (localStorage.getItem("token_access")  === null ||
          localStorage.getItem("token_refresh") === null ) {
        this.$emit("logOut");
        return;
      }
```

```
localStorage.setItem("token_access", "");

await this.$apollo
  .mutate({
    mutation: gql`
      mutation ($refresh: String!) {
        refreshToken(refresh: $refresh) {
          access
        }
      }
    `,
    variables: {
      refresh: localStorage.getItem("token_refresh"),
    },
  })
  .then((result) => {
    localStorage.setItem("token_access", result.data.refreshToken.access);
  })
  .catch((error) => {
    this.$emit("logOut");
    return;
  });

await this.$apollo
  .mutate({
    mutation: gql`
      mutation($transaction: TransactionInput!) {
        createTransaction(transaction: $transaction) {
          date
          id
          usernameDestiny
          usernameOrigin
          value
        }
      }
    `,
    variables: {
      transaction: this.createTransaction,
    },
  })
  .then((result) => {
    alert("Transacción Realizada de Manera Correcta, Revise Historial");
  })
  .catch((error) => {
    alert("Saldo Insuficiente o Destino Incorrecto");
```

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

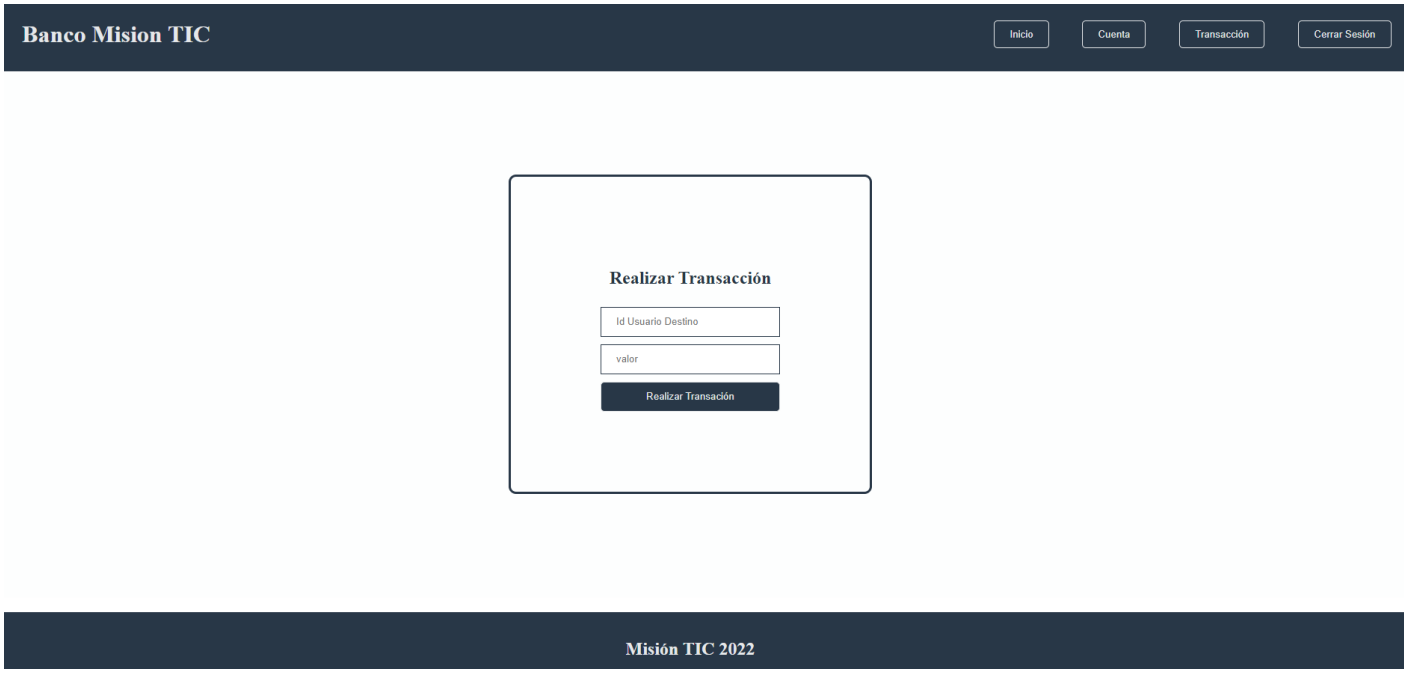
El script está conformado por la definición del objeto que almacenará la información recolectada por el formulario y por la función que procesa el envío del formulario, en esta se definen dos *Mutations*, *refreshToken* y *createTransaction*, y se procesan los resultados de la transacción. Un detalle que se debe tener en cuenta es el uso de las palabras *async* y *await*, pues estas permiten esperar a que la petición *refreshToken* finalice antes de ejecutar la petición *createTransaction*.

El código correspondiente *style* del componente *Transaction.vue* es el siguiente:

```
<style>

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

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

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

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

.transaction h2 {
  color: #283747;
}

.transaction form {
```

```
  width: 50%;
}

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

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

  border: 1px solid #283747;
}

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

  color: #e5e7e9;
  background: #283747;
  border: 1px solid #e5e7e9;

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

.transaction button:hover {
  color: #e5e7e9;
  background: crimson;
  border: 1px solid #283747;
}

</style>
```

Una vez modificado el componente *Transaction.vue*, si se accede a este componente en el navegador web se debe ver lo siguiente:



Conclusiones

El proceso desarrollado en la guía es muy común en los equipos de desarrollo, siempre los sistemas de software evolucionan y se deben agregar y eliminar funcionalidades, de acuerdo con los requisitos del cliente.

Nota: de ser necesario, en el material de la clase se encuentra un archivo llamado *C4a.AP.12. bank_fe.rar*, con el desarrollo del componente realizado en esta guía.