El futuro digital es de todos

MinTIC

Ciclo 3:

Desarrollo de Software

UNIVERSIDAD NACIONAL DE COLOMBIA

MisionTIC 2022

VERSIÓN 1.0

Unidad de educación continua y permanente
Facultad de Ingeniería

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

(57) 1 316 5000
uec_fibog@unaleduco

Componente de Presentación

(Desarrollo Final)

Actividad Práctica

En la sesión anterior se realizó la creación y el desarrollo de algunas funcionalidades del componente de presentación. En esta guía se finalizará el desarrollo del componente.

Desarrollo del Componente

LocalStorage:

En JavaScript se utilizan las siguientes líneas de código para manipular y utilizar el *localStorage* del navegador web:

- Para crear o modificar un item se utiliza *localStorage.setItem(key, value)*, en caso de que la llave no exista se crea el item, si existe se actualiza su valor.
- Para consultar el valor de un item se utiliza *localStorage.getItem(key)*
- Para borrar un item se utiliza *localStorage.removeItem(key)*
- Para borrar todos los items *localStorage.clear()*

Métodos completeLogin y completeSignUp:

El localStorage se utilizará en el desarrollo del componente de presentación para almacenar algunos datos importantes, como los access y refresh tokens, el nombre del usuario y una variable que le permitirá saber a la aplicación en todo momento si el usuario se encuentra o no autenticado.

En la guía anterior se desarrollaron los componentes de inicio de sesión y de registro del usuario, y se crearon en el componente *App.vue* las funciones *completedLogin* y *completedSignUp*, cuyo propósito es procesar las respuestas obtenidas del componente lógico, al realizar la autenticación o el registro correctamente. La idea de dicho procesamiento es almacenar en el localStorage del navegador web la información importante, de forma que esta pueda ser accedida desde cualquier otro componente Vue. Por esta razón, el código de estas dos funciones se debe reemplazar por lo siguiente:

```
completedLogIn: function(data) {
  localStorage.setItem("isAuth", true);
  localStorage.setItem("username", data.username);
  localStorage.setItem("token_access", data.token_access);
  localStorage.setItem("token_refresh", data.token_refresh);
  alert("Autenticación Exitosa");
  this.verifyAuth();
},
```

```
    completedSignUp: function(data) {
      alert("Registro Exitoso");
      this.completedLogIn(data);
    },
```

Algunos detalles a tener en cuenta en estas funciones son los siguientes:

- *isAuth* es un item que no se encuentra en la respuesta del componente lógico, sin embargo, este se almacena para controlar que el usuario se encuentre autenticado, en los componentes que lo requieran.

- La función *alert()* es una función genérica de JavaScript que muestra un diálogo al usuario, con el texto ingresado.

- La respuesta enviada por el componente lógico al iniciar sesión o al registrarse satisfactoriamente, es la misma. Por esta razón, en la función *completedSignUp* se utiliza la función función *completedLogIn*.

Componente Home:

Una vez se han completado las anteriores funciones, se va a crear el componente *Home*, al cual se redireccionará una vez se ha iniciado la sesión. Para ello, se debe crear el archivo *Home.vue* en la carpeta *src/components*, y se debe utilizar el siguiente código:

```
<template>

    <div class="greetings">
        <h1>¡Bienvenido  <span> {{username}} </span>!</h1>
    </div>

</template>


<script>

export default {
    name: "Home",

    data: function(){
        return {
            username: localStorage.getItem('username') || "none"
```

```
        }
    }
}

</script>


<style>
    .greetings{
        margin: 0;
        padding: 0%;
        height: 100%;
        width: 100%;

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

    .greetings h1{
        font-size: 50px;
        color: #283747;
    }

    .greetings span{
        color: crimson;
        font-weight: bold;
    }
</style>
```

Algunos detalles a tener en cuenta en el código del componente son son los siguientes:

- El componente *Home* es un componente muy sencillo, en el que únicamente se muestra un mensaje de bienvenida con el *username* del usuario autenticado.

- El *username* del usuario varía de acuerdo con la persona que se autentica. Por ello, en el *template* se utiliza una sintaxis que permite indicarle a Vue que debe insertar el valor de la variable indicada, en el código HTML (*{ username }*). Esta sintaxis únicamente se puede utilizar en el *template* del componente y es útil para mostrar información variable en la página web. La variable que se indique entre los corchetes (*{}*) debe estar definida en la *data* del componente.

- En la data del componente, a la variable *username* se le asigna el valor almacenado en el item *username* del localStorage. Si este item no existe, se asigna la cadena “none” a la variable.

Registro del componente Home en el Router:

De la misma forma que se explicó en la guía anterior, es necesario registrar los componentes en el router para poder acceder a ellos desde otros componentes. Para registrar al componente *Home*, se debe dirigir al archivo *src/router.js*, importar allí el componente *Home*, y registrarlo en la lista de la variable *routes*. Una vez se ha hecho esto, el código del router 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'
import Home from './components/Home.vue'


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

const router = createRouter({
    history: createWebHistory(),
    routes,
});

export default router;
```

Método verifyAuth:

Como se indicó anteriormente, el componente *Home* será aquel al que se redirija el usuario cuando inicia sesión. Por esta razón, se modificará la función del componente principal *App.vue* que redirecciona al usuario a un componente u otro dependiendo de si está autenticado o no: *verifyAuth*. La idea es que esta función redireccione al usuario al componente *Home* si se encuentra autenticado, o al componente *LogIn* si no lo está. Para ello, se debe reemplazar el código de la función *verifyAuth* (que se encuentra en el archivo *src/App.vue*) por el siguiente:

```
verifyAuth: function() {
      this.is_auth = localStorage.getItem("isAuth") || false;

      if (this.is_auth == false)
        this.$router.push({ name: "logIn" });
      else
        this.$router.push({ name: "home" });
    }
```

Se debe recordar que el *name* utilizando en la función *this.$router.push()* corresponde al nombre asignado al componente al registrarlo en el router. Una vez realizado este cambio, ya es posible que un usuario ingrese a su cuenta utilizando el componente de presentación.

Añadir funcionalidad a los botones Inicio y Cerrar sesión:

Cuando el usuario ingresa a su cuenta, los botones a los que este tiene acceso en la sección superior de la página, cambian. En lugar de mostrar los botones *Iniciar Sesión* y *Registrarse*, se muestran los botones *Inicio*, *Cuenta*, y *Cerrar Sesión*. El propósito del botón *Inicio* es redirigir al usuario al componente *Home*, para ello, se debe crear una función que se encargue de la redirección, y se debe asignar esta función a la etiqueta del botón correspondiente. De esta forma, primero se debe añadir una función a la lista *methods* del componente principal *App.vue*, con el siguiente código:

```
    loadHome: function() {
      this.$router.push({ name: "home" });
    },
```

Y luego, se debe asignar esta función al botón *Inicio*, que se encuentra en el *template* del mismo componente (este botón fue creado en la guía anterior):

```
<button v-if="is_auth" v-on:click="loadHome"> Inicio </button>
```

Por otro lado, el propósito del botón *Cerrar Sesión* es, como su nombre lo indica, cerrar la sesión del usuario. Para ello, cuando se pulse este botón, se eliminarán todos los items almacenados en el localStorage del navegador, y se ejecutará la función *verifyAuth*, que se encargará de redireccionar al usuario a donde corresponda. El código de la función que realizará esto, es el siguiente:

```
    logOut: function () {
      localStorage.clear();
      alert("Sesión Cerrada");
      this.verifyAuth();
    },
```

Para asignar esta función al botón, en el *template* del componente *App.vue* se debe reemplazar la etiqueta del botón *Cerrar Sesión*, por la etiqueta:

```
<button v-if="is_auth" v-on:click="logOut"> Cerrar Sesión </button>
```

Adicionalmente, los componentes hijos del componente principal deben ser capaces de cerrar sesión si algo indebido ocurre, como por ejemplo, el vencimiento del refresh token del usuario. Por esta razón, en el mismo *template* se debe añadir un evento a la etiqueta *router-view*, y asociarle la función *logOut*. Al hacerlo, el código del *router-view* será el siguiente:

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

Componente Account:

La última funcionalidad que se implementará, es mostrar al usuario parte de la información de su cuenta. Esta funcionalidad se desarrollará en el componente *Account*. Sin embargo, antes de implementarla es necesario instalar la dependencia *jwt-decode*, la cual servirá para desencriptar el access token. Para instalarla, se debe

ejecutar el siguiente comando en una terminal ubicada en la carpeta raíz del proyecto:

npm install jwt-decode

Una vez instalada la dependencia, ya se puede implementar el componente *Account*. Para ello, se debe crear el archivo src/components/Account.vue, y se debe utilizar el siguiente código:

```
<template>

    <div v-if="loaded" class="information">
        <h1>Información de su cuenta</h1>
        <h2>Nombre: <span>{{name}}</span></h2>
        <h2>Saldo: <span>{{balance}} COP </span></h2>
        <h2>Correo electrónico: <span>{{email}}</span></h2>
    </div>

</template>


<script>
import jwt_decode from "jwt-decode";
import axios from 'axios';

export default {
    name: "Account",

    data: function(){
        return {
            name: "",
            email: "",
            balance: 0,
            loaded: false,
        }
    },

    methods: {
        getData: async function () {

            if (localStorage.getItem("token_access") === null || localStorage.getItem("toke
n_refresh") === null) {
                this.$emit('logOut');
                return;
            }

            await this.verifyToken();
```

```
            let token = localStorage.getItem("token_access");
            let userId = jwt_decode(token).user_id.toString();

            axios.get(`https://mision-tic-bank-
be.herokuapp.com/user/${userId}/`, {headers: {'Authorization': `Bearer ${token}`}})
                .then((result) => {
                    this.name = result.data.name;
                    this.email = result.data.email;
                    this.balance = result.data.account.balance;
                    this.loaded = true;
                    })
                .catch(() => {
                    this.$emit('logOut');
                });
        },

        verifyToken: function () {
            return axios.post("https://mision-tic-bank-
be.herokuapp.com/refresh/", {refresh: localStorage.getItem("token_refresh")}, {headers: {}}
)
                .then((result) => {
                    localStorage.setItem("token_access", result.data.access);
                })
                .catch(() => {
                    this.$emit('logOut');
                });
        }
    },

    created: async function(){
        this.getData();
    },
}
</script>


<style>
    .information{
        margin: 0;
        padding: 0%;
        width: 100%;
        height: 100%;

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

```
    .information h1{
        font-size: 60px;
        color: #0f1316;
    }

    .information h2{
        font-size: 40px;
        color: #283747;
    }

    .information span{
        color: crimson;
        font-weight: bold;
    }
</style>
```

Algunos detalles a tener en cuenta en el código del componente son son los siguientes:

- Las variables *name*, *email*, y *balance* inician con valores vacíos. El valor real de cada variable se asigna en la función *getData*.

- La variable *loaded* es un booleano que se utiliza para renderizar la información del usuario únicamente cuando ya se ha consultado al componente lógico.

- Cada vez que se muestra el componente *Account* en la pantalla del usuario, se ejecuta la función *created*, la cual ejecuta a su vez a la función *getData*.

- La función *getData* consta de dos secciones, en la primera se verifica que existen los tokens (access y refresh) y se actualiza el access token (utilizando el refresh token); y en la segunda se desencripta el token para conocer el *id* del usuario, y con este se realiza la petición de los datos, ingresando el access token como un header de la petición, para demostrar al componente lógico que el usuario está autenticado.

- En la función *getData*, al recibir la información del usuario, se actualizan las variables del componente, y se indica a Vue que renderice el contenido del componente al cambiar el valor de la variable *loaded*.

- En la función *getData*, si los tokens no existen, si el refresh token ya expiró, o si la petición de los datos del usuario es denegada por el componente lógico, se ejecuta un cierre de sesión.

- La función *getData* es una función *async*, esto le permite utilizar en su definición la palabra reservada *await*. En este caso, esto se utiliza para esperar a que la sección de comprobación y actualización del access token terminen, y que solo cuando hayan terminado, se ejecute la petición de los datos del

usuario. Esta espera es necesaria para evitar hacer una petición de datos con un access token vencido, pues esto resultaría en un cierre de sesión innecesario para el usuario.

Registro del componente Account en el Router:

Una vez creado el componente *Account*, este se debe registrar en el router, de la misma forma que se ha hecho con otros componentes anteriormente. Luego de hacerlo, el código del 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'
import Home from './components/Home.vue'
import Account from './components/Account.vue'


const routes = [{
        path: '/',
        name: 'root',
        component: App
    },
    {
        path: '/user/logIn',
        name: "logIn",
        component: LogIn
    },
    {
        path: '/user/signUp',
        name: "signUp",
        component: SignUp
    },
    {
        path: '/user/home',
        name: "home",
        component: Home
    },
    {
        path: '/user/account',
        name: "account",
        component: Account
    }
];

const router = createRouter({
```

```
    history: createWebHistory(),
    routes,
});

export default router;
```

Método loadAccount:

Finalmente, en el componente principal *App.vue*, se debe asignar al botón *Cuenta* una función que redireccione al componente *Account*. Para ello, se crea la función *loadAccount*:

```
    loadAccount: function () {
      this.$router.push({ name: "account" });
    },
```

Y se asigna al botón *Cuenta*:

```
<button v-if="is_auth" v-on:click="loadAccount"> Cuenta </button>
```

Todo esto, de la misma forma que se ha hecho con componentes anteriores. Con esto ya se ha finalizado el desarrollo de la capa de presentación. Es posible comprobar su funcionamiento ejecutando el servidor con el comando *npm run serve*, e ingresando a la url asignada *http://localhost:8080*.

Nota: de ser necesario, en el material de la clase se encuentra una archivo *C3.AP.16. bank_fe.rar*, con todos los avances en el desarrollo del componente realizado en esta guía.