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 4)

Hasta ahora se han realizado las modificaciones pertinentes al componente web *bank_fe* para adecuarlo a los cambios del sistema desarrollado en este ciclo. Sin embargo, algunos aspectos como la constante actualización del *access token*, no se han manejado, por esta razón, a continuación, se explicarán algunos conceptos y se modificará el proyecto para finalizar el desarrollo del componente web.

Protección de rutas

Hasta ahora la protección de las rutas de la página web se ha manejado utilizando el método *verifyAuth* del componente *App.vue*, este se encarga de redireccionar al usuario a una página u otra dependiendo de si está autenticado o no. Sin embargo, el método *verifyAuth* se utiliza en el *onCreated* del componente, por lo cual, solo se ejecuta en el momento en que se accede a la página web, lo que implica que únicamente se valida y redirige al usuario cuando ingresa a la página web, más no cuando navega por ella.

En una aplicación donde verificar la identidad del usuario es muy importante, se debe asegurar en todo momento que el usuario se encuentra autenticado, no solo en el primer ingreso. Por esta razón, es necesario modificar el proyecto para proteger adecuadamente las rutas de la página web. Uno de los métodos para la protección de rutas es por medio de un *Guard*, es decir, un fragmento de código que se ejecuta antes de cualquier tipo de acceso a una ruta de la página web y determina si el usuario tiene acceso a dicha ruta o no, si no lo tiene, indica la ruta a la cual se debe redirigir al usuario.

Para implementar este *Guard*, el *Router* de Vue.js ofrece un método sencillo para indicar el procesamiento que se debe hacer antes de que el usuario acceda a cualquier ruta. Para ello, justo después de la inicialización del *router*, se utiliza el siguiente código (aún no se debe modificar el proyecto):

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

router.beforeEach((to, from) => {
    return true;
})
```

En este, luego de inicializar el router, se pasa una función como parámetro al método *beforeEach()* del router. Esta función recibe dos parámetros: *to* y *from*, dos objetos de JavaScript que permiten conocer respectivamente las **propiedades** de la ruta **a la que se dirige**, y **desde la que se dirige** el usuario.

Adicionalmente, esta función debe retornar un valor, ya sea un booleano, o una ruta. Si se retorna un booleano, se permite el acceso (*true*) o se deniega (*false*), pero si se retorna una ruta, se redirige al usuario a esta última. De esta forma, en el código anterior se está permitiendo el acceso a cualquier *url* de la página web, desde cualquier *url*.

Existen múltiples formas de reconocer una ruta (*to* o *from*) de acuerdo con sus propiedades, o de retornar una ruta en la función, sin embargo, el método más sencillo es utilizando los nombres (*name*) que se le otorga a cada ruta en el arreglo *routes* del archivo *src/router.js*. Para reconocer una ruta que se recibe como parámetro se puede acceder a la propiedad *name* de cualquiera de los dos parámetros recibidos:

```
    to.name;
    from.name;
```

Por otro lado, para retornar una ruta se puede retornar un objeto que contenga la propiedad *name* de la ruta a la que se desea redirigir al usuario:

```
    return { name: "logIn" };
```

De esta forma, si se desea, por ejemplo, permitir únicamente el acceso a las rutas con el nombre *logIn* y *signUp*, y que cualquier otra ruta sea redirigida a la ruta *logIn*, se puede utilizar el siguiente código:

```
router.beforeEach((to, from) => {
    if (to.name == "logIn" || to.name == "signUp") {
        return true;
    } else {
        return { name: "logIn" };
    }
})
```

Preprocesamientos antes de acceder a una ruta

Un *Guard* no sirve solo para proteger rutas, en estos se puede hacer algún procesamiento de datos que se deba realizar justo antes de acceder a cualquier ruta (o a ciertas rutas de acuerdo con lo que se defina). Por ejemplo, si se necesita almacenar algún valor importante en el *localStorage* antes de que el usuario acceda a cualquier ruta, se puede utilizar el siguiente código:

```
router.beforeEach((to, from) => {
    localStorage.setItem("Key", "Value");
```

```
    return true;
})
```

Metadatos de las rutas

Adicionalmente, se pueden definir metadatos para cada una de las rutas, para que en el *Guard* o en los componentes de Vue.js, se realice una u otra acción de acuerdo con su valor. Estos se suelen utilizar, por ejemplo, para indicar que rutas requieren o no de autenticación para ser accedidas por el usuario.

Para definir metadatos a una ruta, basta con dirigirse a la especificación de la ruta (en el proyecto sería en el arreglo *routes* del archivo *src/router.js*), y añadir la propiedad *meta*, cuyo valor sea un objeto con cada uno de los metadatos. Un ejemplo sería el siguiente:

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

Para acceder a los metadatos desde un Guard se utiliza el código:

```
to.meta.requiresAuth;
from.meta.requiresAuth;
```

Mientras que para acceder a los metadatos desde un componente de Vue.js se utiliza el código:

```
this.$route.meta.requiresAuth;
```

Con este último se accede a los **metadatos** de la **ruta en la que se encuentra actualmente** el componente.

Modificaciones del Router

Con estos conceptos ya se pueden hacer los cambios necesarios al *Router* para implementar el *Guard* que requiere el componente web. Para ello, se debe reemplazar el código del archivo *src/router.js* por lo siguiente:

```
import gql from "graphql-tag";
import { createRouter, createWebHistory } from "vue-router";
import { ApolloClient, createHttpLink, InMemoryCache } from '@apollo/client/core'

import LogIn from './components/LogIn.vue'
import SignUp from './components/SignUp.vue'
import Home from './components/Home.vue'
import Account from './components/Account.vue'
import Transaction from './components/Transaction.vue'

const routes = [{
        path: '/user/logIn',
        name: "logIn",
        component: LogIn,
        meta: { requiresAuth: false }
    },
    {
        path: '/user/signUp',
        name: "signUp",
        component: SignUp,
        meta: { requiresAuth: false }
    },
    {
        path: '/user/home',
        name: "home",
        component: Home,
        meta: { requiresAuth: true }
    },
    {
        path: '/user/account',
        name: "account",
        component: Account,
        meta: { requiresAuth: true }
    },
    {
        path: '/user/transaction',
        name: "transaction",
        component: Transaction,
        meta: { requiresAuth: true }
```

```
    }
];

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

const apolloClient = new ApolloClient({
    link: createHttpLink({ uri: 'https://mision-tic-api-gateway.herokuapp.com/' }),
    cache: new InMemoryCache()
})

async function isAuth() {
    if (localStorage.getItem("token_access") === null ||
localStorage.getItem("token_refresh") === null) {
        return false;
    }

    try {
        var result = await apolloClient.mutate({
            mutation: gql `
                mutation ($refresh: String!) {
                    refreshToken(refresh: $refresh) {
                        access
                    }
                }
            `,
            variables: {
                refresh: localStorage.getItem("token_refresh"),
            },
        })

        localStorage.setItem("token_access", result.data.refreshToken.access);
        return true;
    } catch {
        localStorage.clear();
        alert("Su sesión expiró, por favor vuelva a iniciar sesión");
        return false;
    }
}

router.beforeEach(async(to, from) => {
    var is_auth = await isAuth();

    if (is_auth == to.meta.requiresAuth) return true
```

```
    if (is_auth) return { name: "home" };
    return { name: "logIn" };
})

export default router;
```

Para comprender los cambios realizados al código, se deben tener en cuenta lo siguiente:

- A cada una de las rutas se añade un metadato que permitirá saber si en dicha ruta se requiere que el usuario esté autenticado o no.

- Se importan las dependencias *ApolloClient*, *createHttpLink*, y *InMemoryCache*, y se crea un cliente de *Apollo*, que se utiliza en la función *isAuth()*.

- Se crea la función *isAuth()*, la cual verifica si el usuario está autenticado. Si lo está, se hace la petición correspondiente al API Gateway para actualizar el *access token* y se retorna *true*, si no lo está, se retorna *false*.

- En la función *isAuth()* se utiliza un *try-catch*, en lugar de un *.then()-.catch()* porque desde esta última no se pueden retornar valores. Sin embargo, su funcionamiento es exactamente el mismo.

- En la creación del *Guard*, se utiliza la función *isAuth()* para saber si el usuario está autenticado y de acuerdo con esto, permitir o redirigir la ruta dependiendo de lo indicado por los metadatos de la ruta a la que intenta acceder el usuario.

Un detalle importante que se debe tener en cuenta, es que cada vez que se accede a una ruta con el *access token* y el *refresh token* almacenados en el *localStorage*, se actualiza el *access token* del usuario, pues de esta forma se verifica que su sesión no ha caducado. Además, esto permite realizar futuras peticiones con el *access token* sin inconvenientes.

Modificaciones del componente App.vue

Una vez se ha implementado el *Guard*, se debe modificar el *script* del componente *App.vue*, pues ya no es necesario utilizar la función *verifyAuth*, y se debe cambiar la forma en la que se obtiene y modifica la variable *is_auth*. Para ello, el código que debe utilizar en este componente es el siguiente:

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

  computed: {
    is_auth: {
```

```
    get: function() {
      return this.$route.meta.requiresAuth;
    },
    set: function() { }
  }
},

methods:{

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

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

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

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

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

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

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

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

```
    },
  }
}
</script>
```

Para comprender los cambios realizados al código, se deben tener en cuenta lo siguiente:

- Se eliminó el método *verifyAuth*, pues como se mencionó anteriormente, el *Guard* desarrollado se encargará de verificar las rutas dependiendo del estado de la sesión del usuario.

- Se eliminó la propiedad *created* del componente, pues ya no se va a ejecutar ninguna función al acceder al componente.

- Se eliminó la propiedad *data* del componente, pues la única variable definida allí se obtendrá de otra manera.

- Se definió la propiedad *computed* dentro del componente, dentro de esta se definen las variables computadas del componente, es decir, las variables que no son estáticas (como las definidas en la propiedad *data*), cuyo valor depende del resultado de la función *get* definida en el objeto asociado al nombre de la variable. Una característica importante de estas variables, y la razón por la cual se está utilizando, es que Vue.js por su cuenta evalúa constantemente la función para saber si el valor ha cambiado y actualizar la variable si lo ha hecho.

- Se definió la variable computada *is_auth*, cuyo valor se actualizará dinámicamente cada vez que se cambie de ruta.

Nota: de ser necesario, en el material de la clase se encuentra un archivo llamado *C4a.AP.13. bank_fe.rar*, con el desarrollo del componente realizado en esta guía.