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 1)

Para desarrollar el componente web del sistema de software planteado se utilizará como base el proyecto del componente front-end desarrollado en el ciclo 3: *bank_fe*. Por ello, a continuación se realizarán las configuraciones necesarias para adaptar dicho proyecto a una arquitectura de microservicios y para añadir las nuevas funcionalidades del sistema.

Nota: de ser necesario, en el material de la clase se encuentra un archivo llamado *C4a.AP.10. bank_fe_inicial.rar*, con el componente desarrollado en Ciclo 3. Es necesario recordar que antes de ejecutar el proyecto se debe utilizar el comando *npm install* para instalar todas las dependencias.

Archivo HTML base y metadatos

En el ciclo 3 se abordaron aspectos muy importantes de Vue.js, como la creación de componentes, la conexión con una API REST, el enrutamiento de urls a componentes, entre otros. Sin embargo, hubo un concepto básico del funcionamiento de Vue.js que no se profundizó, y es que la integración de componentes parte de un documento HTML, desde el cual se puede modificar distintos aspectos como, por ejemplo, los metadatos de la página web.

Dicho documento HTML base se encuentra en la carpeta *public* con el nombre *index.html*. Su contenido es el siguiente:

index.html

bank_fe > public > index.html > html > body > div#app

```
<!DOCTYPE html>
<html lang="">

  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title>
      <%= htmlWebpackPlugin.options.title %>
    </title>
  </head>

  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript enabled.
          Please enable it to continue.</strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>

</html>
```

En este se definen el *header* y el *body* de la página web. En este último se ubican 2 etiquetas principales, la etiqueta *noscript*, que sirve para mostrar un mensaje al usuario en caso de que JavaScript no esté habilitado en su navegador; y la etiqueta *div* vacía, cuyo *id* es *app*. Este *id* es aquel que se utiliza en la función *mount* del archivo *src/main.js* para buscar y reemplazar el *div* adecuado, por el componente indicado por Vue.js:

main.js

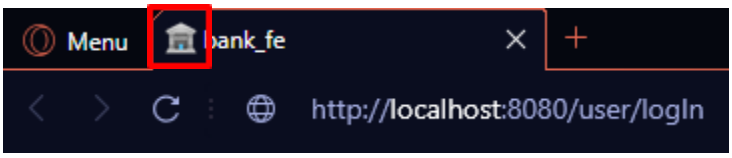
bank_fe > src > main.js

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'

createApp(App).use(router).mount('#app')

```

Por otro lado, en el header se definen los metadatos de la página web. Para modificar estos, basta con asignar los atributos adecuados a las etiquetas que se desean cambiar. Por ejemplo, para cambiar el ícono que se muestra en la pestaña del navegador, simplemente se debe eliminar el archivo *public/favicon.ico*, y se debe ubicar dentro de la carpeta *public/* una imagen *.ico* con el ícono que se desea mostrar. En este caso, se utilizará la imagen de un banco con el nombre *favicon.ico* para no cambiar el nombre de la

referencia al ícono en el archivo *public/index.html*, sin embargo, si se desea utilizar otro nombre, se debe tener en cuenta dicho cambio en el atributo *href* de la etiqueta *link*.

Para observar el cambio, se debe ejecutar el servidor con el comando *npm run serve*, y abrir la aplicación web en el navegador. En la pestaña se debe ver el nuevo ícono:



Instalación de paquetes

En el ciclo 3 se conectó el componente web a la capa lógica del sistema por medio de una interfaz REST. En el sistema de software actual, el componente web se debe conectar a un API Gateway por medio de una interfaz GraphQL, por lo cual, se utilizará un cliente GraphQL. Dicho cliente se implementará en próximas sesiones, sin embargo, antes de hacerlo se deben instalar en el proyecto todas las dependencias requeridas, utilizando el comando *npm install graphql graphql-tag apollo-link-context vuex @vue/apollo-composable @apollo/client @vue/apollo-option*:

```
d:\bank_fe>npm install graphql graphql-tag apollo-link-context vuex @vue/apollo-composable @apollo/client @vue/apollo-option
[...................] | loadIdealTree:loadAllDepsIntoIdealTree: sill install loadIdealTree
```

Adicionalmente, dado que ya no se va a realizar ninguna conexión REST, se puede eliminar la dependencia *axios*, la cual se utilizaba con este propósito. Para ello, se debe ejecutar el comando *npm uninstall axios*:

```
d:\bank_fe>npm uninstall axios
[                .....] \ postinstall: sill doSerial postinstall 32
```

Configuración inicial del proyecto

Una vez instaladas todas las dependencias necesarias, se debe configurar el proyecto para que los componentes de Vue.js tengan acceso a las diferentes herramientas, como el cliente de Apollo. Para ello, se debe reemplazar el código del archivo *src/main.js*, por el siguiente código:

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import { ApolloClient, createHttpLink, InMemoryCache } from '@apollo/client/core'
import { createApolloProvider } from '@vue/apollo-option'
```

```
import { setContext } from 'apollo-link-context'

const httpLink = createHttpLink({
    uri: 'https://mision-tic-api-gateway.herokuapp.com/',
})

const authLink = setContext((_, { headers }) => {
    return {
        headers: {
            ...headers,
            "Authorization": localStorage.getItem("token_access") || ""
        }
    }
})

const apolloClient = new ApolloClient({
    link: authLink.concat(httpLink),
    cache: new InMemoryCache()
})

const apolloProvider = new createApolloProvider({
    defaultClient: apolloClient
})

createApp(App).use(router).use(apolloProvider).mount('#app')
```

En este código se mantiene la misma estructura para la creación del servidor que se manejaba anteriormente, sin embargo, se añade alguna lógica adicional. Dentro de lo añadido, se importan todas las herramientas de *apollo* necesarias para configurar el cliente GraphQL y se inicializan para utilizar el cliente dentro de los componentes del proyecto.

En esta configuración se inicializan cuatro herramientas y se almacenan en variables. Cada una de las herramientas tiene un propósito:

- *createHttpLink*: Permite crear un link *http* a partir de una cadena (*string*), que podrá ser interpretado por otras funciones o herramientas de JavaScript. **Es importante que se cambie la cadena asociada con la *uri*, por la *url* del API Gateway del proyecto personal desplegado en sesiones anteriores**.

- *setContext*: Crea un contexto a partir de un objeto, el cual podrá ser interpretado por otras funciones o herramientas para añadir dicho contexto a las peticiones HTTP. En este caso, se añade a los headers de la petición que utilice este contexto, el header *"Authorization"*, cuyo valor será el *access token* almacenado en el *localStorage* **al momento de hacer la petición**, si en ese momento este

dato no se encuentra en el *localStorage*, el valor del header *"Authorization"* será una cadena vacía (*""*).

- *apolloClient*: Crea un cliente GraphQL de Apollo, a partir de las configuraciones indicadas. En dichas configuraciones se asigna el tipo de caché, el link *http*, y el *context* que el cliente utilizará cada vez que haga una petición. Este cliente se puede utilizar para realizar peticiones al API Gateway, sin embargo, ya que se van a realizar peticiones al API Gateway desde los componentes de Vue.js, es necesario crear un *Apollo Provider*.

- *createApolloProvider*: Crea un *Apollo Provider* a partir de un *apolloClient*. Esto es importante, porque por defecto, los componentes de *Vue.js* no tienen acceso al cliente de Apollo, por lo cual, el *Apollo Provider* se encarga de hacer las configuraciones necesarias para que dicho cliente pueda ser accedido desde cualquier componente del proyecto.

Finalmente, una vez se han realizado todas estas configuraciones, se le debe indicar a Vue.js que utilice el *Apollo Provider*, para ello se utiliza la función *.use(apolloProvider)*. Algo importante con respecto a la utilización de la función *.use()*, es que esta se encarga además de exponer una variable global a todos los componentes del proyecto. Para acceder a una variable global, desde cualquier componente se puede utilizar la sintaxis *this.$varName*, donde *varName* varía de acuerdo con la variable global a la que se desea acceder.

Cada variable global dispone de distintas funcionalidades de acuerdo con el objeto ingresado en la función *.use()*, por ejemplo, en el caso del *.use(router)* se puede acceder a la variable global *this.$router*, desde la cual se pueden utilizar funcionalidades para manejar las rutas de la página web; mientras que en el caso del *.use(apolloProvider)* se puede acceder a la variable global *this.$apollo*, esto se debe tener en cuenta, pues en próximas sesiones se utilizará para hacer peticiones al API Gateway.

Nota: de ser necesario, en el material de la clase se encuentra un archivo llamado *C4a.AP.10. bank_fe_final.rar*, con el desarrollo del componente realizado en esta guía.