	UNIVERSIDAD DON BOSCO **FACULTAD DE INGENIERÍA** **ESCUELA DE COMPUTACIÓN**	
CICLO: 02	**GUIA DE LABORATORIO #08**	
	Nombre de la Práctica:	**React , parte II**
	MATERIA:	**Diseño y Programación de Software Multiplataforma**

I. OBJETIVOS

Que el estudiante:

- Diseñe aplicaciones web utilizando funciones de React.
- Diseñe aplicaciones con navegación.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Aprender a crear componente personalizado.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Aprender a utilizar pipes dentro de la interfaz de usuario.
- Hacer uso de los componentes en el diseño de interfaz React.

Contenido

II. INTRODUCCION TEORICA

Utilizar Axios con React

Axios es un cliente HTTP que funciona en el navegador o del lado del servidor con node.js. Es una de las librerías más populares y cuenta con algunas características que la API Fetch no proporciona por defecto como son los interceptors, convierte la respuesta del servidor a JSON automáticamente y también proporciona protección contra ataques cross-site request forgery (XSRF).

Para empezar, necesitamos instalar axios mediante npm con el siguiente comando:

npm install axios

Después en nuestro proyecto podemos crear un nuevo archivo llamado http.service.js y dentro de este archivo vamos a escribir el siguiente código:

```
import axios from 'axios';
// Se crea instancia http con valores default
const httpInstance = axios.create( {
   baseURL: 'https: //www.api.com/'
});
```

En esta sección de código solo importamos la librería de axios y creamos una instancia que nos permitirá utilizar ciertos valores por default que llevaran todas nuestras peticiones, en nuestro caso es la URL del servidor. Para más información sobre todos los valores que acepta axios.
Podemos utilizar un método especial de axios llamado interceptors que nos ayuda a capturar las peticiones o respuestas.

```
// Interceptor para enviar datos recibidos y checar errores
httpInstance.interceptors.response.use(null, error => {
   const expectedError = error.response && error.response.status >= 400 && error.response.status
< 500;
   if (!expectedError) {
      // Loggear mensaje de error a un servicio como Sentry
      // Mostrar error genérico al usuario
      return Promise.reject(error);
   }
);

export default httpInstance;
```

En esta sección verificamos si la respuesta nos arrojó un error y si este error se encuentra del rango de errores esperados (errores del cliente) y en caso de ser un error inesperado (errores del servidor

con status >= 500) podemos mostrar algún mensaje genérico al usuario y hacer un log de este tipo de errores. Por ultimo reenviamos la respuesta del servidor al bloque de código que realizo la petición y exportamos nuestra instancia de axios.

Para realizar peticiones simplemente tenemos que importar la instancia de axios que creamos en el archivo http.service.js de la siguiente forma:

```
import http from './http.service';
```

Luego podemos hacer una petición de la siguiente forma:

```
http.get('usuarios').then(usuarios => {
   console.log(usuarios);
}).catch(error => {
   console.error(error);
})
```

Conclusión

Hacer peticiones mediante axios nos ahorra algunas líneas de código además de proporcionarnos características útiles como los interceptors.

No te olvides de revisar la documentación de axios para conocer más detalles.

https://github.com/axios/axios

React Hooks, useEffect.

Añadiendo funcionalidad en el ciclo de vida de nuestro componente

Ya sabemos qué son los hooks y cómo usar el hook **useState** para añadir un estado a nuestro componente, para que puedan tener comportamiento y sean dinámicos. vamos a conocer el **hook useEffect** que, sin duda, será otro de los hooks más utilizados. ¿Su función? **Ejecutar código cada vez que nuestro componente se renderiza.**

useEffect: accediendo al ciclo de vida de nuestro componente

El ciclo de vida de los componentes en React permitía en nuestros componentes con class poder ejecutar código en diferentes **fases de montaje, actualización y desmontaje.** De esta forma, podíamos añadir cierta funcionalidad en las distintas etapas de nuestro componente.

Con los hooks también podremos acceder a ese ciclo de vida en nuestros componentes funcionales aunque de una forma más clara y sencilla. Para ello usaremos **useEffect, un hook que recibe como parámetro una función que se ejecutará cada vez que nuestro componente se renderice, ya sea por un cambio de estado, por recibir props nuevas o, y esto es importante, porque es la primera vez que se monta.**

Para usar este hook, primero debemos importarlo desde la librería de React.

```
import React, { useEffect } from 'react'
```

Ahora, en nuestro componente funcional, **vamos a añadir un efecto que se ejecutará cada vez que nuestro componente se renderice.** Para eso, ejecutaremos el método **useEffect** dentro del cuerpo de nuestra función y le pasaremos como parámetro la función que queremos que ejecute al renderizar el componente.

```
import React, { useEffect } from 'react'

function Example() {
  useEffect(function () {
    console.log('render!')
  })

  return <span>This is a useEffect example</span>
}
```

¡Esto hará que se muestre en consola el mensaje render! después que el componente se renderice por primera vez. Por si te lo estás preguntando, en este ejemplo, el método useEffect ha funcionado de forma similar a como lo hubiera hecho el ciclo de vida componentDidMount:

```
import React, { Component } from 'react'

class Example extends Component {
  componentDidMount () {
    console.log('render!')
  }

  render () {
    return (<span>This is a componentDidMount example</span>)
  }
}
```

useEffect: usando el state de nuestro componente y creando un efecto

Ahora que ya hemos usado **useEffect** vamos a utilizarlo junto con el hook **useState** que ya conocemos.. Para ello, vamos a recuperar el ejemplo del Contador pero vamos a hacer que, cada vez que se vaya a renderizar de nuevo el componente, actualice el título de la página con un mensaje indicando el número de veces que hemos hecho click en el botón. **Para ello tendremos que leer el valor actual del estado interno de nuestro componente de la siguiente forma**:

```
import React, { useEffect, useState } from 'react'

function Contador() {
 const [count, setCount] = useState(0)

 useEffect(() => {
  // Actualiza el title de la página en cada click!
  document.title = `Has hecho clic ${count} veces`
 })

 return (
  <div>
   <span>El contador está a {count}</span>
   <button onClick={() => setCount(count + 1)}>
    Incrementar contador
   </button>
  </div>
 )
}
```

Para verlo en funcionamiento, podéis acceder a la demo desde vuestro navegador. Así podréis comprobar que el título de la página se actualiza:

- Nada más entrar en la página**.** Ya que se ejecuta **useEffect** al montarse nuestro componente.
- Cada vez que hacemos click en el componente. Cuando el **state** cambia, esto dispara un nuevo renderizado y, al renderizarse de nuevo, se vuelve a ejecutar la función que le hemos pasado a useEffect.

  Para seguir con las comparaciones con las clases, en este ejemplo nuestro useEffect está funcionando como el ciclo de vida **componentDidMount** y como el ciclo de vida **componentDidUpdate**. Esto nos ayuda a ver que pese a la sencillez que parece tener este hook, esconde un gran potencial.

Ejemplo práctico:

1. Crear nuestro primer Proyecto: **create-react-app crudreactphp**
2. Instalar los paquetes **npm i bootstrap reactstrap axios**
   (Bootstrap -> estilo css
   Reactstrap -> ventanas modales https://reactstrap.github.io/
   Axios -> realizar peticiones HTTP)
3. Adjunto repositorio del código fuente:
   https://github.com/AlexanderSiguenza/crudreactphp.git
4. La explicación estará en un video que les compartiré en discord
5. La pantalla debería verse de la siguiente forma;

V. DISCUSION DE RESULTADOS

1. Deberán de replicar el siguiente ejercicio, pero para la tabla **productos (id, nombre, existencias) , debe de estar en un hosting.**

VII. BIBLIOGRAFIA

- React, una biblioteca de JavaScript. (2013-2020). Facebook, Inc. Jordan Walke. Recuperado de https://es.reactjs.org/docs/getting-started.html