ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DISEO DE INTERFACES (TDSD322)

ASIGNATURA:	Diseño de Interfaces
PROFESOR:	Ing. Yadyra Franco
PERÍODO ACADÉMICO:	2024-B

ESTUDIANTE:

- Jhonny Villanueva M.

INVESTIGACIÓN Y ANÁLISIS

1. Uso de `export` en React

En React, el **`export`** permite exportar funciones, variables y componentes para utilizarlos en otros archivos o módulos dentro de tu proyecto. Esto ayuda a mantener el código modular y facilita la reutilización, permitiendo un mejor mantenimiento y escalabilidad de tu aplicación.

Existen dos formas principales de exportar en JavaScript y React:

Exportación por defecto (**`default export`**)
Exportación con nombre (**`named export`**)

1.1. Ejemplo de exportación por defecto

```jsx
// Archivo: Header.jsx
import React from 'react';

function Header() {
  return <h1>Bienvenido a mi aplicación</h1>;
}

export default Header;
```

1.2. Para importar el componente Header en otro archivo:

```jsx
// Archivo: App.jsx
import Header from './Header';

function App() {
  return (
    <div>
      <Header />
    </div>
  );
}

export default App;
```

1.3. Ejemplo de exportación con nombre:

```jsx
// Archivo: utils.js
export const suma = (a, b) => a + b;
export const resta = (a, b) => a - b;
```

1.4. Para importar estas funciones en otro archivo:

```jsx
// Archivo: App.jsx
import { suma, resta } from './utils';

console.log(suma(5, 3)); // Resultado: 8
console.log(resta(5, 3)); // Resultado: 2
```

¿Por qué usar **`export`** en React?

- **Modularidad**: Divide el código en piezas reutilizables y fáciles de entender.
- **Reutilización**: Evita duplicar código al reutilizar funciones y componentes en distintos archivos.
- **Mantenimiento**: Facilita el mantenimiento y la depuración al tener código más ordenado.

2. Reutilización de componentes y funciones

React permite reutilizar componentes y funciones gracias a **`export`**. Esto es fundamental para crear aplicaciones más organizadas y eficientes.

Ejemplo práctico de reutilización de un componente

```jsx
// Archivo: Button.jsx
import React from 'react';

export function Button({ label, onClick }) {
  return <button onClick={onClick}>{label}</button>;
}
```

Este componente **`Button`** puede reutilizarse en múltiples partes de la aplicación:

```jsx
// Archivo: App.jsx
import React from 'react';
import { Button } from './Button';

function App() {
  const handleClick = () => alert('Botón clicado');

  return (
    <div>
      <Button label="Haz clic aquí" onClick={handleClick} />
      <Button label="Enviar" onClick={() => console.log('Enviado')} />
    </div>
  );
}

export default App;
```

3. Análisis de la función en el archivo de la caja

Supongamos que tenemos una función en el archivo **`caja.js`**:

```jsx
// Archivo: caja.js
import React, { useState } from 'react';

function Caja({ inicial }) {
  const [contador, setContador] = useState(inicial);

  const incrementar = () => setContador(contador + 1);
  const decrementar = () => setContador(contador - 1);

  return (
    <div>
      <h2>Contador: {contador}</h2>
      <button onClick={incrementar}>Incrementar</button>
      <button onClick={decrementar}>Decrementar</button>
    </div>
  );
}

export default Caja;
```

Explicación línea por línea

1. **`import React, { useState } from 'react';`**
   Importa React y el hook useState para manejar el estado del contador.

2. **`function Caja({ inicial }) {`**
   Declara un componente funcional Caja que recibe una prop inicial.

3. **`const [contador, setContador] = useState(inicial);`**
   Crea un estado contador con su función para actualizarlo (setContador), inicializado con el valor de inicial.

4. **`const incrementar = () => setContador(contador + 1);`**
   Define una función para incrementar el contador en 1.

5. **`const decrementar = () => setContador(contador - 1);`**
   Define una función para decrementar el contador en 1.

6. **`return (...)`**
   Devuelve el JSX que muestra el valor del contador y dos botones para incrementar o decrementar.

4. Respuestas a preguntas

- **¿Dónde utilizar JSX?**

  JSX se utiliza dentro de los componentes de React para describir la estructura de la interfaz de usuario. Por ejemplo:

```
function App() {
  return <h1>Hola, mundo</h1>;
}
```

- **¿Por qué usar JSX y no solo HTML en React?**

Sintaxis declarativa: JSX permite escribir código similar a HTML dentro de JavaScript, facilitando la creación de componentes complejos.

Integración con JavaScript: Puedes utilizar expresiones JavaScript directamente en JSX.

Optimización y seguridad: JSX se compila a llamadas **React.createElement()**, optimizando el rendimiento.

- **Diferencias entre JSX y HTML**

Característica	JSX	HTML
Clases CSS	`className="boton"`	`class="boton"`
Atributos	`htmlFor="inputId"`	`for="inputId"`
Expresiones JS	`{valor}`	`No se permiten expresiones JS`
Cierre de etiquetas	`<img />`	`<img>`

- **¿Qué es React?**

React es una biblioteca de JavaScript para construir interfaces de usuario. Fue desarrollada por Facebook y se utiliza para crear aplicaciones web de una sola página (SPA) mediante un sistema de componentes reutilizables.

- **¿Cuándo apareció React?**

React fue lanzado en **mayo de 2013** por Facebook.

Buenas prácticas en React

1. **Componentes pequeños y reutilizables:** Divide tu aplicación en componentes modulares.
2. **Nombra los componentes con mayúsculas:** Ejemplo: MiComponente.
3. **Evita la lógica compleja en el renderizado:** Utiliza hooks o funciones auxiliares para manejar lógica.
4. **Usa propTypes o TypeScript:** Para validar las props de los componentes.
5. **Gestiona el estado con cuidado:** Utiliza useState para estados locales y useReducer o bibliotecas como Redux para estados globales.
6. **Mantén un formato consistente:** Utiliza herramientas como Prettier y ESLint para formatear el código automáticamente.

BIBLIOGRAFIA

1. B. B. McLaughlin, JavaScript: The Definitive Guide. Sebastopol, CA, USA: O'Reilly Media, 2020.
2. React, React Blog, Dic. 2024. [Online]. Available: https://legacy.reactjs.org/tutorial/tutorial.html
3. K. Simpson, You Don’t Know JS: ES6 & Beyond. Sebastopol, CA, USA: O'Reilly Media, 2016.
4. J. Duckett, HTML & CSS: Design and Build Websites. Hoboken, NJ, USA: Wiley, 2011.
5. J. L. Carmack and J. D. Smith, "Component-based UI design in React," in Proceedings of the 12th International Conference on Web Engineering (ICWE), Berlin, Germany, 2017, pp. 180–190.