

EJERCICIOS PRACTICA REACT

Enunciado del ejercicio 3. – 6.

Crear una aplicación React que muestre un contador implementado mediante un componente funcional y un componente de clase, utilizando props para pasar datos y state para gestionar la información interna.

Ejercicio React: Components, Props y State

Componente Funcional

Usuario: Carlos

Contador: 8

Incrementar

Componente de Clase

Usuario: María

Contador: 5

Incrementar

SOLUCION:

[CounterFunctional.js](#)

```
import { useState } from 'react';

// COMPONENTE FUNCIONAL
// Se define como una función JavaScript
function CounterFunctional(props) {

    // DESTRUCTURING DE PROPS
    // Se extrae la prop 'name' enviada desde el componente padre
    const { name } = props;

    // STATE CON HOOK useState
    // count: valor del estado
    // setCount: función para actualizar el estado
    const [count, setCount] = useState(0);

    // JSX
    // HTML dentro de JavaScript

    return (
        <div>
            {/* COMPONENTES */}
            {/* Este es un componente React reutilizable */}
            <h2>Componente Funcional</h2>

            {/* PROPS */}
            <p>Usuario: {name}</p>

            {/* STATE */}
            <p>Contador: {count}</p>

            {/* EVENTO + setState */}
            <button onClick={() => setCount(count + 1)}>
                Incrementar
            </button>
        </div>
    );
}

// EXPORTACIÓN DEL COMPONENTE
export default CounterFunctional;
```

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

function CounterFunctional(props) {
  const { name } = props;
  const [count, setCount] = useState(0);

  return (
    <div>
      <h2>Componente Funcional</h2>
      <p>Usuario: {name}</p>
      <p>Contador: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Incrementar
      </button>
    </div>
  );
}

export default CounterFunctional;
```

CounterFunctional.js

```
import { Component } from 'react';

// COMPONENTE DE CLASE
// Se define usando una clase que extiende Component
class CounterClass extends Component {

    // CONSTRUCTOR
    // Inicializa el estado y recibe las props
    constructor(props) {
        super(props);

        // STATE
        // El estado se define como un objeto
        this.state = {
            count: 0
        };
    }

    // METODO PARA ACTUALIZAR EL ESTADO
    // Usa setState (forma clasica)
    increment = () => {
        this.setState({ count: this.state.count + 1 });
    };

    // METODO RENDER
    // Devuelve el JSX que se muestra en pantalla
    render() {
        // DESTRUCTURING DE PROPS Y STATE
        const { name } = this.props;
        const { count } = this.state;
    }
}
```

```

    ...
    return (
      <div>
        <h2>Componente de Clase</h2>

        {/* PROPS */}
        <p>Usuario: {name}</p>

        {/* STATE */}
        <p>Contador: {count}</p>

        {/* EVENT HANDLING */}
        <button onClick={this.increment}>
          Incrementar
        </button>
      </div>
    );
}

// EXPORTACIÓN DEL COMPONENTE
export default CounterClass;

```

App.js

```

import CounterClass from './CounterClass';
import CounterFunctional from './CounterFunctional';

function App() {
  return (
    <div>
      <h1>Ejercicio React: Components, Props y State</h1>

      <CounterFunctional name="Carlos" />
      <CounterClass name="María" />
    </div>
  );
}

export default App;

```

 **Enunciado del ejercicio 7. – 10.**

En este ejercicio vas a crear una pequeña aplicación en React que permita añadir números a una lista.

Cada vez que el usuario pulse un botón, se añadirá un nuevo número a la lista y se mostrará en pantalla.

La lista debe renderizarse usando map y cada elemento debe tener una key correcta, evitando usar el índice del array como clave.

Si la lista está vacía, se mostrará un mensaje indicándolo.

La aplicación debe tener un estilo básico con CSS.

Listado de Números

Añadir número

No hay números en la lista

App.js

```
import { useState } from 'react';
import './styles.css';

function App() {
  const [numbers, setNumbers] = useState([]);

  const addNumber = () => {
    const newNumber = {
      id: Date.now(), // key correcta
      value: numbers.length + 1
    };

    setNumbers([...numbers, newNumber]);
  };

  return (
    <div className="container">
      <h1>Lista de números</h1>

      <button onClick={addNumber}>
        Añadir número
      </button>

      {numbers.length === 0 && (
        <p>No hay números en la lista</p>
      )}

      <ul>
        {numbers.map(number => (
          <li key={number.id}>
            Número: {number.value}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

Styles.css

```
.container {  
    width: 300px;  
    margin: 40px auto;  
    text-align: center;  
    font-family: Arial, sans-serif;  
}  
  
button {  
    padding: 8px 15px;  
    margin-bottom: 10px;  
    cursor: pointer;  
}  
  
li {  
    list-style: none;  
    margin: 5px 0;  
    padding: 5px;  
    background: #f2f2f2;  
}
```

 **Enunciado del ejercicio 11. – 15.**

Crea un formulario donde el usuario ingrese nombres.

Al enviar, los nombres se muestran en una lista y el input recibe automáticamente el foco.

Formulario de nombres

Ingresa tu nombre	Agregar
-------------------	---------

No hay nombres agregados aún

```
import { useState, useRef } from 'react';
import './styles.css';

function App() {
  const [name, setName] = useState('');
  const [names, setNames] = useState([]);

  // 15. Ref para el input
  const inputRef = useRef(null);

  // Manejo del formulario
  const handleSubmit = (e) => {
    e.preventDefault();
    if (name.trim() === '') return;

    setNames([...names, { id: Date.now(), value: name }]);
    setName('');

    // 15.1 Focus automático usando ref
    inputRef.current.focus();
  };

  return (
    <h1>Formulario de nombres</h1>
    /* 11. Manejo de formulario */
    <form onSubmit={handleSubmit}>
      <input
```

```
    <input  
      ref={inputRef} // 15.. Ref  
      type="text"  
      value={name}  
      onChange={(e) => setName(e.target.value)}  
      placeholder="Ingresa tu nombre"  
    />  
    <button>Agregar</button>  
  </form>  
  
  /* 8.. Renderizado condicional */  
  {names.length === 0 ? (  
    <p>No hay nombres agregados aún</p>  
  ) : (  
    <ul>  
      {names.map((n) => (  
        <li key={n.id}>{n.value}</li> // 9.1 Key única  
      ))}  
    </ul>  
  )}  
  </div>  
};  
  
export default App;
```

Styles.css

```
body {
    font-family: Arial, sans-serif;
}

form {
    margin-bottom: 15px;
}

input {
    padding: 6px 10px;
    margin-right: 5px;
}

button {
    padding: 6px 10px;
    cursor: pointer;
}

ul {
    list-style: none;
    padding: 0;
}

li {
    margin: 5px 0;
    background: #f0f0f0;
    padding: 5px;
}
```

```
import { useState } from 'react';
import './styles.css';

function App() {
  const [numbers, setNumbers] = useState([]);

  const addNumber = () => {
    const newNumber = {
      id: Date.now(), // ✅ key correcta
      value: numbers.length + 1
    };

    setNumbers([...numbers, newNumber]);
  };

  return (
    <div className="container">
      <h1>Lista de Números</h1>

      {/* 7. Event Handling */}
      <button onClick={addNumber}>
        Añadir número
      </button>

      {/* 8. Conditional Rendering */}
      {numbers.length === 0 && (
        <p>No hay números en la lista</p>
      )}
    
```

```
.... /* 9.. List Rendering */
.... <ul>
....   {numbers.map(number => (
....     <li key={number.id}>
....       Número: {number.value}
....     </li>
....   )));
.... </ul>
.... </div>
.... );
}

export default App;
```

Styles.css

```
.container {
  width: 300px;
  margin: 40px auto;
  text-align: center;
  font-family: Arial, sans-serif;
}

button {
  padding: 8px 15px;
  margin-bottom: 10px;
  cursor: pointer;
}

li {
  list-style: none;
  margin: 5px 0;
  padding: 5px;
  background: #f2f2f2;
}
```

 **Enunciado del ejercicio 16. – 21.**

Crea un botón que abra un **modal** usando Portals.

Cuando se abra, hace una petición a la API de usuarios y muestra solo el nombre del primer usuario.

Si hay un error, se muestra un mensaje con **Error Boundary**.

Modal de Usuario



App.js

```
import { useState } from 'react';
import ErrorBoundary from './ErrorBoundary';
import UserModal from './UserModal';

function App() {
  const [showModal, setShowModal] = useState(false);

  return (
    <div>
      <h1>Modal de Usuario</h1>
      {/* 7. Event Handling: onClick */}
      <button onClick={() => setShowModal(true)}>Abrir Modal</button>

      {/* 17. Error Boundary */}
      {showModal && (
        <ErrorBoundary>
          {/* 16. Portals */}
          <UserModal onClose={() => setShowModal(false)} />
        </ErrorBoundary>
      )}
    </div>
  );
}

export default App;
```

UserModal.js

```

import { useEffect, useState } from 'react';
import ReactDOM from 'react-dom';

function UserModal({ onClose }) {
  const [user, setUser] = useState(null);

  useEffect(() => {
    // 1.1. HTTP GET Request
    fetch('https://jsonplaceholder.typicode.com/users/1')
      .then(res => res.json())
      .then(data => setUser(data))
      .catch(() => {
        throw new Error('Error cargando usuario'); // 1.7. Error Boundary
      });
  }, []);

  // 1.6. Portals: renderizamos fuera del root
  return ReactDOM.createPortal(
    <div style={{ background: '#fff', padding: '20px', border: '1px solid #000' }}>
      {/* 7. Event Handling */}
      <button onClick={onClose}>Cerrar</button>

      {/* 8. Conditional Rendering */}
      {user ? <p>Nombre: {user.name}</p> : <p>Cargando...</p>}
    </div>, document.getElementById('portal-root')
  );
}

export default UserModal;

```

ErrorBoundary.js

```
import { Component } from 'react';

class ErrorBoundary extends Component {
  state = { hasError: false };

  // 12. Component Lifecycle Methods: Mounting / Updating
  static getDerivedStateFromError() {
    return { hasError: true }; // 17. Error Boundary
  }

  render() {
    if (this.state.hasError) return <p>An error occurred</p>; // 17. Error Boundary
    return this.props.children;
  }
}

export default ErrorBoundary;
```

Styles.css

```
body {
  font-family: Arial, sans-serif;
  text-align: center;
  margin: 40px;
}

button {
  padding: 6px 12px; /* 7. Event Handling: clickable button */
  cursor: pointer;
  margin-top: 10px;
}
```