Lesson #1: ReactJS Fundamentals

Objetivo

Comprender los conceptos básicos de ReactJS, su propósito, características principales y diferencias con otros frameworks o librerías.

Teoría

¿Qué es ReactJS?

Es una librería de JavaScript para construir interfaces de usuario interactivas y eficientes.

Desarrollada por Facebook (Meta) y lanzada en 2013.

Se enfoca en la capa de vista (UI) en aplicaciones web.

Características principales:

Virtual DOM para optimizar actualizaciones.

Arquitectura basada en componentes reutilizables.

Programación declarativa.

JSX:

Sintaxis que permite escribir HTML dentro de JavaScript.

Mejora la legibilidad y productividad del código.

Práctica

Crea un archivo App.js con el siguiente contenido:

```jsx
import React from 'react';
function App() {
  return ¡Hola, React!;
}
```

```
export default App;
```

Explica qué hace este componente y cómo se renderiza en una aplicación React.

Preguntas para Auto Estudio

Esta lección cubre preguntas como:

- ¿Qué es ReactJS?
- ¿Quién desarrolló ReactJS?
- ¿Es React un framework o una librería?
- ¿Cuáles son sus características principales?
- ¿Qué es JSX y por qué se usa?

Lesson #2: Componentes en React

Objetivo

Aprender qué son los componentes en React, los tipos existentes y cómo crearlos y reutilizarlos.

Teoría

Componentes: Bloques reutilizables de UI.

Tipos:

Funcionales (function components)

De clase (class components)

Props: Permiten pasar datos de un componente padre a uno hijo.

PropTypes y defaultProps: Ayudan a validar y definir valores por defecto para las props.

Práctica

Crea un componente funcional llamado Saludo que reciba una prop nombre y muestre un mensaje personalizado.

jsx

```
function Saludo({ nombre }) {
  return <h2>Hola, {nombre}!</h2>;
}
```

Usa este componente dentro de App.js:

jsx

```
<Saludo nombre="Juan" />
```

Preguntas para Auto Estudio

Esta lección cubre preguntas como:

- ¿Qué son los componentes?

- ¿Cuáles son los tipos de componentes?

- ¿Qué son las props?

- ¿Qué son defaultProps y propTypes?

Lesson #3: Estado y Ciclo de Vida en React

Objetivo

Comprender el manejo del estado (state) y el ciclo de vida de los componentes en React.

Teoría

State:

Es un objeto local y mutable que almacena datos dinámicos.

En componentes de clase: this.state y this.setState().

En componentes funcionales: hook useState.

Ciclo de vida:

Métodos especiales en componentes de clase: componentDidMount, componentWillUnmount, etc.

En componentes funcionales: hook useEffect.

Práctica

Crea un componente funcional con un contador usando useState.

```jsx
import React, { useState } from 'react';
function Contador() {
  const [count, setCount] = useState(0);
  return (

Contador: {count}
 setCount(count + 1)}>Incrementar

  );
```

}

```

Explica cómo se actualiza el estado y cómo se refleja en la UI.

Preguntas para Auto Estudio

Esta lección cubre preguntas como:

- ¿Qué es el state?

- ¿Cómo se inicializa y actualiza el state?

- ¿Qué son los métodos de ciclo de vida?

- ¿Cómo se usa useEffect?
```

Lesson #4: Hooks en React

Objetivo

Comprender qué son los hooks, su propósito y cómo utilizar los hooks más comunes en componentes funcionales.

Teoría

¿Qué son los hooks?

Son funciones que permiten a los componentes funcionales usar estado y otras características de React.

Introducidos en React 16.8.

Principales hooks:

useState: Manejo de estado local.

useEffect: Manejo de efectos secundarios (side effects).

useContext: Acceso a contextos globales.

useRef, useMemo, useCallback, useReducer.

Reglas de los hooks:

Solo se llaman en el nivel superior de componentes funcionales o custom hooks.

No se llaman en ciclos, condicionales ni funciones anidadas.

Práctica

Crea un componente que use useEffect para mostrar un mensaje en consola cada vez que cambie el estado:

```jsx
import React, { useState, useEffect } from 'react';
function Mensaje() {
```

```
  const [count, setCount] = useState(0);
  useEffect(() => {
    console.log('El contador cambió:', count);
  }, [count]);
  return (
     setCount(count + 1)}>
      Incrementar ({count})

  );
}
```

Ejercicio: Crea un custom hook llamado useContador que permita incrementar y decrementar un valor.

jsx

```
import { useState } from 'react';
function useContador(valorInicial = 0) {
  const [valor, setValor] = useState(valorInicial);
  const incrementar = () => setValor(v => v + 1);
  const decrementar = () => setValor(v => v - 1);
  return { valor, incrementar, decrementar };
}
```

Ejemplo: Usa useContext para compartir un color entre componentes.

jsx

```
import React, { createContext, useContext } from 'react';

const ColorContext = createContext('blue');

function ColorBox() {
  const color = useContext(ColorContext);
  return <div style={{ background: color, width: 100, height: 100 }} />;
}

function App() {
  return (
    <ColorContext.Provider value="red">
      <ColorBox />
    </ColorContext.Provider>
  );
}
```

Preguntas para Auto Estudio

Esta lección cubre preguntas como:

- ¿Qué son los hooks?
- ¿Cuándo se introdujeron?
- ¿Para qué sirve useState, useEffect, useContext, etc.?
- ¿Cuáles son las reglas de los hooks?

Lesson #5: React Avanzado (Context, Redux, Fragments, HOCs)

Objetivo

Explorar conceptos avanzados de React como Context, Redux, Fragments y Higher-Order Components (HOCs).

Teoría

Context: Permite compartir datos globalmente sin prop drilling.

Redux: Librería para manejo de estado global predecible.

Fragments: Agrupan elementos sin añadir nodos extra al DOM.

HOCs: Funciones que reciben un componente y devuelven un nuevo componente con funcionalidades extra.

Práctica

Crea un contexto simple y consúmelo en un componente hijo usando useContext.

jsx

```
import React, { createContext, useContext } from 'react';
const UserContext = createContext('Invitado');
function Saludo() {
    const usuario = useContext(UserContext);
    return <h2>Hola, {usuario}</h2>;
}
function App() {
    return (
        <UserContext.Provider value="Ana">
            <Saludo />
```

```
        </UserContext.Provider>
    );
}
```

Ejercicio: Implementa un contador global usando Redux Toolkit (puedes consultar la documentación oficial para detalles).

Ejemplo: Usa un fragmento para devolver múltiples elementos sin un div extra.

jsx

```
function Lista() {
    return (
        <>
            <li>Elemento 1</li>
            <li>Elemento 2</li>
        </>
    );
}
```

Ejercicio: Crea un Higher-Order Component que agregue un borde a cualquier componente recibido.

jsx

```
function withBorder(Component) {
```

```
    return function(props) {
        return <div style={{ border: '2px solid black' }}><Component {...props} /></div>;
    };
}
```

Preguntas para Auto Estudio

Esta lección cubre preguntas como:

- ¿Qué es Context y para qué sirve?
- ¿Qué es Redux?
- ¿Qué son los Fragments?
- ¿Qué es un Higher-Order Component?

Lesson #6: Routing, Formularios y Componentes Controlados

Objetivo

Aprender a manejar rutas, formularios y la diferencia entre componentes controlados y no controlados en React.

Teoría

React Router: Permite navegación entre vistas sin recargar la página.

Formularios controlados: El valor de los inputs es gestionado por el estado de React.

Formularios no controlados: El valor de los inputs es gestionado por el DOM.

Práctica

Instala React Router y crea dos rutas simples (/ y /about).

jsx

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
function Home() { return <h2>Inicio</h2>; }
function About() { return <h2>Acerca de</h2>; }
function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Inicio</Link> | <Link to="/about">Acerca de</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
```

```
            </Routes>
        </BrowserRouter>
    );
}
```

Crea un formulario controlado con un input de texto y un botón.

jsx

```
import React, { useState } from 'react';
function Formulario() {
    const [valor, setValor] = useState('');
    const handleSubmit = e => {
        e.preventDefault();
        alert('Valor enviado: ' + valor);
    };
    return (
        <form onSubmit={handleSubmit}>
            <input value={valor} onChange={e => setValor(e.target.value)} />
            <button type="submit">Enviar</button>
        </form>
    );
}
```

Crea un formulario no controlado usando ref.

jsx

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

```
function FormNoControlado() {
    const inputRef = useRef();
    const handleSubmit = e => {
        e.preventDefault();
        alert('Valor enviado: ' + inputRef.current.value);
    };
    return (
        <form onSubmit={handleSubmit}>
            <input ref={inputRef} />
            <button type="submit">Enviar</button>
        </form>
    );
}
```

Ejercicio: Crea una validación simple en el formulario controlado para no permitir enviar si el campo está vacío.

Preguntas para Auto Estudio

Esta lección cubre preguntas como:

- ¿Para qué sirve React Router?
- ¿Qué es un componente controlado?
- ¿Qué es un componente no controlado?
- ¿Cómo se maneja un formulario en React?

Lesson #7: Testing y Performance en React

Objetivo

Conocer las herramientas y técnicas para testear componentes y optimizar el rendimiento en aplicaciones React.

Teoría

Testing:

Herramientas: Jest, React Testing Library.

Shallow rendering vs. full rendering.

React.StrictMode para detectar problemas.

Performance:

Lazy loading de componentes.

Memoización (useMemo, React.memo, useCallback).

Code splitting.

Server Side Rendering (SSR) e hidratación.

Práctica

Escribe un test simple para un componente usando React Testing Library.

jsx

```
// Componente a testear
function Saludo({ nombre }) {
  return <h1>Hola, {nombre}</h1>;
}
// Test
import { render, screen } from '@testing-library/react';
```

```
test('muestra el nombre', () => {
  render(<Saludo nombre="Juan" />);
  expect(screen.getByText('Hola, Juan')).toBeInTheDocument();
});
```

Implementa lazy loading de un componente con React.lazy y Suspense.

```
jsx
import React, { Suspense, lazy } from 'react';
const OtroComponente = lazy(() => import('./OtroComponente'));
function App() {
  return (
    <Suspense fallback={<div>Cargando...</div>}>
      <OtroComponente />
    </Suspense>
  );
}
```

Usa useMemo para optimizar un cálculo costoso.

```
jsx
import React, { useMemo, useState } from 'react';
function CalculoCostoso({ valor }) {
  const resultado = useMemo(() => {
    // Simula un cálculo pesado
    let total = 0;
    for (let i = 0; i < 1000000; i++) {
```

```
      total += valor * Math.random();
    }
    return total;
  }, [valor]);
  return <div>Resultado: {resultado}</div>;
}
```

Ejercicio: Escribe un test para verificar que un botón incrementa un contador.

Ejemplo: Usa React.StrictMode en el entry point de tu app para detectar problemas potenciales.

```
jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Preguntas para Auto Estudio

Esta lección cubre preguntas como:

- ¿Cómo se testean los componentes?

- ¿Qué es shallow rendering?

- ¿Qué es React.StrictMode?

- ¿Qué es lazy loading?

- ¿Qué es memoización?

- ¿Qué es SSR e hidratación?