**Clase 15:**  Introducción a React.js

Objetivos de la clase

* Entender qué es React.js y su función en el desarrollo de interfaces de usuario.
* Aprender los conceptos fundamentales de React, incluyendo componentes y estado.
* Configurar un entorno de desarrollo para trabajar con React.

**Preguntas orientadoras para la lectura**

* *¿Qué es React.js y cuáles son los principios fundamentales que lo distinguen de otras bibliotecas o frameworks para el desarrollo de interfaces de usuario?*
* *¿Cómo se configura un entorno de desarrollo para trabajar con React.js utilizando herramientas como Node.js, npm y Create React App, y cuál es la estructura básica de un proyecto React?*
* *¿Cuál es la diferencia entre componentes funcionales y componentes de clase en React, y cómo se crean y utilizan estos componentes en una aplicación React?*
* *¿Qué es el estado en React, cómo se gestiona en los componentes de clase, y cómo influye en el renderizado condicional y la interacción del usuario?*

**Introducción a React.js**

*Definición*

React.js es una biblioteca de JavaScript desarrollada por Facebook para construir interfaces de usuario interactivas y dinámicas. Su enfoque principal es la creación de vistas en aplicaciones web, facilitando la construcción de interfaces complejas mediante la reutilización de componentes.

A diferencia de otros enfoques, React se centra en el rendimiento y la eficiencia al actualizar solo las partes del DOM que han cambiado, lo que resulta en una experiencia de usuario más fluida y rápida.

|  |
| --- |
| **Importante:** en el aula encontrarás el documento asociado a **DOM (Document Object Model)** para poder profundizar y conocer más. |

*Principios fundamentales de React*

* **Componentes:** React se basa en la construcción de interfaces a partir de componentes reutilizables. Cada componente es una pieza independiente de la interfaz que puede tener su propia lógica y estilo.
* **Estado (State):** El estado en React permite que los componentes mantengan datos que pueden cambiar con el tiempo, y actualiza la vista en respuesta a esos cambios.
* **Propiedades (Props):** Las propiedades son datos que se pasan de un componente padre a un componente hijo, permitiendo la configuración y personalización de los componentes.
* **JSX (JavaScript XML):** JSX es una extensión de sintaxis para JavaScript que permite escribir código similar a HTML dentro de JavaScript. Facilita la creación de componentes y la definición de la estructura de la interfaz.

*Ventajas de utilizar React.js*

* **Reutilización de componentes:** La modularidad y reutilización de componentes permite un desarrollo más ágil y mantenible.
* **Virtual DOM:** React utiliza un Virtual DOM para minimizar las actualizaciones directas al DOM real, mejorando el rendimiento.
* **Unidirectional Data Flow:** La gestión de datos en React sigue un flujo unidireccional, lo que facilita el seguimiento de los cambios y la depuración.

**Configuración del entorno de desarrollo**

*Instalación de Herramientas*

* **Node.js y npm:** Node.js es el entorno de ejecución de JavaScript que permite la ejecución del código de React en el servidor. npm (Node Package Manager) es la herramienta utilizada para gestionar las dependencias del proyecto, incluyendo React y otras bibliotecas necesarias.
* **Create React App:** Create React App es una herramienta de línea de comandos que simplifica la configuración inicial de un proyecto React. Automatiza la configuración del entorno de desarrollo y proporciona una estructura de proyecto predeterminada.

*Creación de un nuevo proyecto React*

Para crear un nuevo proyecto React, se utiliza el comando **npx create-react-app nombre-del-proyecto**, que inicializa un nuevo proyecto con una configuración predeterminada. Esto incluye una estructura básica de archivos y una serie de herramientas de desarrollo preconfiguradas.

*Estructura básica de un proyecto React*

* **public/**: Contiene archivos estáticos como el archivo **index.html**, que sirve como punto de entrada de la aplicación.
* **src/**: Contiene el código fuente de la aplicación, incluidos los archivos de componentes, estilos y otros recursos. El archivo principal es `index.js`, que se encarga de renderizar el componente raíz de la aplicación.
* **package.json**: Archivo de configuración que gestiona las dependencias del proyecto y define scripts útiles para el desarrollo.

**Componentes en React**

*Conceptos*

Los componentes son bloques fundamentales en React que encapsulan la lógica, la estructura y el estilo de una parte de la interfaz de usuario. Los componentes pueden ser funcionales o de clase, y cada uno tiene su propia forma de manejar el estado y las propiedades.

*Creación y renderizado de componentes*

* **Componentes funcionales:** Son funciones de JavaScript que retornan JSX. Son más simples y recomendados para componentes que no necesitan manejar estado o ciclo de vida.

Ejemplo:

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**Componentes de Clase:** Son clases de JavaScript que extienden de **React.Component** y tienen métodos de ciclo de vida y manejo del estado.

Ejemplo:

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

*Uso de Propiedades (Props)*

Las propiedades permiten pasar datos desde un componente padre a un componente hijo. Esto permite la personalización y reutilización de componentes.

Ejemplo:

function App() {

return <Bienvenida name="Laura" />;

}

**Estado en React**

*Conceptos*

El estado en React permite a los componentes gestionar datos que pueden cambiar con el tiempo y afectar la representación de la interfaz de usuario. Cada componente puede tener su propio estado, y cualquier cambio en el estado provocará una re-renderización del componente.

*Uso del estado en componentes de clase*

En componentes de clase, el estado se define en el constructor y se actualiza mediante el método **setState()**.

Ejemplo:

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

increment = () => {

this.setState({ count: this.state.count + 1 });

}

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

*Actualización del estado y renderizado condicional*

El estado se actualiza utilizando **setState()**, lo que provoca una re-renderización del componente con el nuevo estado. El renderizado condicional permite mostrar diferentes partes de la interfaz en función del estado.

Ejemplo:

render() {

return (

<div>

{this.state.count % 2 === 0 ? <p>Even</p> : <p>Odd</p>}

</div>

);

}