

Teoría React I

¿Qué es una biblioteca?

Una biblioteca (o librería) es un conjunto de herramientas y funciones predefinidas que los desarrolladores pueden utilizar para agilizar el desarrollo de sus aplicaciones y mejorar la eficiencia de su código.

En este contexto, nos centraremos en librerías escritas en **JavaScript**.

¿Qué es React?

React es una biblioteca de JavaScript que facilita la creación de interfaces de usuario interactivas y eficientes mediante el uso de componentes reutilizables y un enfoque eficiente para actualizar el DOM.

Es especialmente popular en el desarrollo de **aplicaciones de una sola página (SPA)** y ha sido adoptado por muchas empresas y desarrolladores debido a su simplicidad y rendimiento.

React organiza la interfaz de usuario en piezas pequeñas que llamamos **componentes**. Cada componente puede tener su propia información (también llamado **estado** interno) y se puede componer con otros componentes para construir interfaces más complejas.

React utiliza un **Virtual DOM** para mejorar el rendimiento. En lugar de actualizar directamente el DOM (Document Object Model), React realiza cambios en una representación virtual del DOM y luego calcula las diferencias (diffing) para actualizar solo las partes necesarias del DOM real. Esto minimiza las manipulaciones directas en el DOM, lo que mejora la eficiencia y la velocidad de la aplicación.

¿Qué es Vite?

Vite es un entorno de desarrollo para crear aplicaciones web basado en JavaScript y TypeScript. Fue creado para optimizar el proceso de desarrollo al proporcionar una experiencia de desarrollo altamente eficiente y rápida.

Vite crea en pocos segundos la estructura de archivos y carpetas necesarias

para que funcione correctamente una aplicación de React!

¿JavaScript o TypeScript?

La diferencia principal entre JavaScript (JS) y TypeScript (TS) radica en la forma en que manejan la tipificación. La **tipificación** se refiere al concepto de especificar qué tipo de valor puede almacenar una variable.

- JavaScript (JS): es un lenguaje de programación que se ejecuta en el navegador web, es de tipado dinámico ya que las variables pueden cambiar de tipo durante la ejecución del programa. Al ser interpretado, los errores de tipo a menudo se descubren durante la ejecución.
- TypeScript (TS): es un superset de JS, es decir, JS con capacidades adicionales como el tipado estático ya que las variables se declara con el tipo de dato y el compilador de TypeScript verifica estos tipos durante el tiempo de compilación, ayudando a detectar errores de tipo antes de que el código se ejecute.

¿Qué son los componentes de React?

Las aplicaciones de React están hechas a partir de componentes. Un componente es una pieza de UI (siglas en inglés de interfaz de usuario) que tiene su propia lógica y apariencia. Un componente puede ser tan pequeño como un botón, o tan grande como toda una página.

Los componentes pueden ser declarados como funciones o como clase pero este curso se enfocará en trabajar con componentes funcionales, es decir, funciones que devuelven **markup** (marcado o etiquetas):

```
Unset
export default function MiTitulo() {
  return (
    <h2 className="text-red">Titulo de la sección</h2>
  );
}
```

Ahora que has declarado MyButton, puedes anidarlo en otro componente de la siguiente manera:

Unset

```
import MiTitulo from "../ruta/archivo"

export default function Bienvenida() {

  return (

    <>

      <MiTitulo />

    </>

  );

}
```

Nota que `<MiTitulo />` empieza con mayúscula. Los nombres de los componentes de React siempre deben comenzar con mayúscula, mientras las etiquetas HTML deben estar minúsculas.

La declaración `export default` se utiliza al crear módulos de JavaScript para exportar por defecto funciones, objetos o tipos de dato primitivos del módulo para que puedan ser utilizados por otros programas con la sentencia `import`.

Al dividir una aplicación en componentes, se facilita el mantenimiento y la reutilización del código. Además, los componentes en React pueden actualizarse de manera eficiente gracias al Virtual DOM, lo que mejora el rendimiento general de la aplicación.

¿Qué es JSX/TSX?

La sintaxis de marcado que viste arriba se llama JSX/TSX. Es totalmente opcional, pero la mayoría de los proyectos de React usan esta extensión por la comodidad que ofrece. Todas las herramientas que recomendamos para el desarrollo local son compatibles con JSX/TSX sin ningún tipo de configuración.

JSX/TSX combina HTML y TypeScript/JavaScript en un solo archivo, lo que facilita la creación de componentes de React y la representación de la interfaz de usuario UI.

JSX/TSX es más estricto que HTML: - Los nombres de los archivos y de las

funciones que definen componentes de React deben escribirse con **PascalCase**.
- Tienes que cerrar etiquetas como `
`, ``, `<input />`, `<Componente />` y `<CualquierComponente />`. - Como es una función, tu componente tampoco puede devolver múltiples etiquetas de JSX (debe devolver una única "cosa"). Debes envolverlas en un padre compartido, como `<div>...</div>` o en un envoltorio vacío `<>...</>`. - Tienes que definir clases con el atributo `className` en lugar de `class`. - El `export default` habilitará el uso de una etiqueta `<Componente />` para poder utilizarse donde sea necesario.

¿Cómo mostrar datos en un componente?

Las llaves `{}` en JSX/TSX son un elemento poderoso que te permite "volver" a **JavaScript** desde el contexto del marcado (etiquetas). También puedes «escaparte hacia JavaScript» en los atributos implementando llaves en lugar de comillas.

```
Unset
import MiTitulo from "../ruta/archivo"

export default function Bienvenida() {
  const subtitulo = "Subtitulo de la vista"
  const rutaImagen = "bienvenida.png"
  return (
    <>
      <MiTitulo />
      <h4>{subtitulo}</h4>
      <img src={rutaImagen} />
    </>
  );
}
```

¿Cómo realizar un renderizado condicional?

En algunas situaciones necesitarás renderizar algo bajo ciertas condiciones. En

React, no hay una sintaxis especial para escribir condicionales. En cambio, usarás las mismas técnicas que usas al escribir código regular de JS/TS. Podrás utilizar la sentencia `if`:

```
Unset
export default function Bienvenida() {

  let mensaje

  if (edad > 18) {

    mensaje = "puedes ingresar"

  } else {

    mensaje = "debes ser mayor"

  }

  return (

    <>

      {mensaje}

    </>

  );
}
```

Si prefieres un código más compacto, puedes utilizar el operador `?` condicional. A diferencia de `if`, funciona dentro de JSX:

```
Unset
export default function Bienvenida() {

  return (

    <>

      {(edad > 18) ? ("puedes ingresar") : ("debes ser mayor")}

    </>

  );
}
```

```
}
```

Cuando no necesites la rama `else`, puedes también usar la sintaxis lógica `&&`, más breve:

```
Unset
<div>

  {isLoggedIn && <AdminPanel />}

</div>
```

¿Cómo pasar información entre componentes?

Los componentes de React utilizan props para comunicarse entre sí. Cada componente padre puede enviar información a sus componentes hijos mediante el uso de props. Las props pueden parecerte similares a los atributos HTML, pero permiten pasar cualquier valor de JavaScript a través de ellas, como objetos, arrays y funciones.

Las `props` son los datos que se pasan a un elemento JSX. Por ejemplo, `className`, `src`, `alt`, `width` y `height` son algunas de las props que se pueden pasar a un elemento ``.

Las props que puedes utilizar con cualquier etiqueta HTML están predefinidas. Sin embargo, puedes definir y pasar cualquier `prop` a tus propios componentes para personalizarlos.

```
Unset
<MiTitulo

  titulo = "Mostrar este titulo"

  color = "rojo"

  usuario = { nombre: "nombre" }

/>
```

Luego, el componente `MiTitulo` recibe estos atributos o propiedades como un objeto que es necesario desestructurar para poder usar esas `props`. Puedo

desestructurar de dos formas:

Unset

```
export default function MiTitulo(props) {  
  const { titulo,color }= props  
  return ( );  
}  
  
export default function MiTitulo({ titulo,color }) {  
  return ( );  
}
```

Utiliza la sintaxis que mejor entiendas! Considera las props como «controles» que puedes ajustar. Cumplen el mismo papel que los argumentos de una función—de hecho, ¡las props son el único argumento de tu componente! Las funciones de los componentes de React aceptan un único argumento, un objeto props.

¿Cuáles son las ventajas de pasar props?

El pasaje de `props` (abreviatura de "propiedades") en React es un concepto fundamental y esencial para entender cómo se construyen y se manejan las interfaces de usuario en esta biblioteca.:

- **Reutilización de Componentes:** El uso de `props` permite que los componentes de React sean reutilizables. Puedes crear un componente genérico y luego pasarlo diferentes `props` para modificar su comportamiento o apariencia en diferentes contextos. Esto fomenta la reutilización del código y reduce la redundancia.
- **Mantenimiento y Escalabilidad:** Al utilizar `props`, los componentes de React se vuelven más fáciles de mantener y escalar. Puedes modificar un componente en un solo lugar, y esos cambios se reflejarán en todos los lugares donde se utilice el componente, siempre que las `props` relevantes se pasen de manera adecuada.
- **Control de Flujo de Datos:** React sigue un patrón de flujo de datos unidireccional, donde las `props` se pasan de componentes padres a hijos. Esto hace que el flujo de datos sea más predecible y fácil de rastrear, lo que a su vez ayuda en la depuración y comprensión del funcionamiento

de la aplicación.

- **Personalización y Configuración:** Las `props` permiten personalizar componentes sin tener que modificar el código del componente en sí. Por ejemplo, un componente `Button` podría aceptar `props` como `color`, `size` y `onClick`, permitiendo que se use en varios contextos con diferentes estilos y funcionalidades. En resumen, el pasaje de `props` es fundamental en React porque facilita la reutilización de componentes, mejora el mantenimiento y la escalabilidad de las aplicaciones, y permite una comunicación efectiva y un flujo de datos claro entre componentes.

¿Qué es la componentización?

El proceso de componentización en React es fundamental para construir aplicaciones eficientes y mantenibles. Consiste en descomponer la interfaz de usuario en elementos más pequeños y manejables (componentes). El proceso de componentización es iterativo y evolutivo. Conforme tu aplicación crezca y cambie, probablemente tendrás que volver a visitar y refinar tus componentes. La clave es mantener los componentes lo suficientemente flexibles para ser reutilizados, pero también lo suficientemente específicos para ser útiles en su contexto particular.

1. **Identificar Componentes:** Primero, analiza la interfaz de tu aplicación para identificar distintos elementos que pueden ser independientes. Piensa en términos de reutilización y responsabilidades únicas. Por ejemplo, botones, tarjetas de producto, barras de navegación, y formularios son buenos candidatos para convertirse en componentes.
2. **Estructura de Archivos:** Organiza tus componentes en una estructura de archivos lógica (carpeta `components` y carpeta `/views`).
3. **Definir Props:** Define las `props` para cada componente. Las `props` son los parámetros que pasas a un componente para darle funcionalidad y datos. Piensa en las `props` como en los argumentos de una función: deben permitir que el componente realice su trabajo sin exponer innecesariamente detalles internos.
4. **Construir la Interfaz de Usuario con Componentes:** Comienza a construir tu interfaz de usuario utilizando estos componentes. Reemplaza los elementos HTML con tus componentes React donde sea apropiado. Por ejemplo, en lugar de usar directamente un elemento `<button>` en tu JSX/TSX, podrías usar un componente `<Button>` que has creado.
5. **Refactorizar y Optimizar:** Refactoriza y optimiza tus componentes a medida que avanza tu proyecto. Puedes encontrar oportunidades para dividir un componente grande en varios más pequeños, o tal vez descubras que ciertos componentes pueden fusionarse.

6. **Reutilización y Mantenimiento:** Considera la reutilización de componentes en diferentes partes de tu aplicación. Un buen componente es aquel que puede utilizarse en múltiples situaciones sin modificaciones o con mínimas adaptaciones.

¿Qué son los CSS modules?

Los módulos de CSS en React, comúnmente llamados "CSS Modules", son una forma de encapsular estilos en módulos para evitar conflictos y mejorar el mantenimiento del código en proyectos de desarrollo web. Los CSS Modules se utilizan para definir estilos específicos para componentes individuales, asegurando que estos estilos no afecten a otros componentes en la aplicación.

Aquí hay algunos conceptos clave sobre los CSS Modules en el contexto de React:

1. **Encapsulación de estilos:** Los CSS Modules permiten que los estilos se apliquen de manera local a un componente específico en lugar de tener un alcance global. Cada componente puede tener sus propios estilos sin preocuparse por conflictos con otros componentes.
2. **Nombres únicos:** Cuando utilizas CSS Modules, los nombres de clases en tus archivos CSS se vuelven únicos para cada componente. Esto se logra mediante la asignación automática de nombres únicos a las clases CSS durante el proceso de construcción. El sistema de construcción (como Webpack) se encarga de la asignación de nombres únicos.
3. **Sintaxis de importación:** En React, los estilos CSS Modules se importan de la misma manera que otros módulos de JavaScript.

Unset

```
import styles from './MiComponente.module.css';
```

4. **Acceso a estilos en componentes:** Los estilos importados a través de CSS Modules se acceden como un objeto en el componente React. Cada clase definida en el archivo CSS se convierte en una propiedad de ese objeto.



Unset

```
// Ejemplo de uso de estilos en un componente React
```

```
<div      className={styles.miClase}>Contenido      del  
componente</div>
```

5. **Preprocesadores:** Puedes utilizar preprocesadores CSS como Sass o Less junto con CSS Modules para aprovechar sus características adicionales, como variables y anidamiento.

El uso de CSS Modules facilita la modularidad y el mantenimiento del código en aplicaciones grandes o en equipos de desarrollo colaborativos, ya que reduce la posibilidad de conflictos de estilos. Al mismo tiempo, sigue siendo compatible con el ecosistema existente de React y puede integrarse fácilmente con las herramientas de construcción comunes como Webpack.

💡 Como sugerencia, resulta beneficioso recurrir a fuentes oficiales de información al enfrentar dudas o al buscar una comprensión más profunda sobre algún tema. A continuación, te proporcionamos enlaces de utilidad para que puedas investigar a tu propio ritmo en el momento que lo consideres oportuno:  [React](#)  [Vite](#)