BOOTCAMP

Generación T

streambe

Componentes, JSX y Props

Los bloques fundamentales para crear aplicaciones modernas con React. En esta segunda clase, exploraremos las herramientas esenciales para construir interfaces de usuario interactivas y reutilizables.

Objetivos de la Clase 2

Entender qué es JSX

Aprender la sintaxis, diferencias con HTML y sus ventajas.

Comprender los Componentes

Descubrir por qué son la base de React y cómo reutilizarlos.

Aprender sobre las Props

Entender su importancia para la comunicación entre componentes.

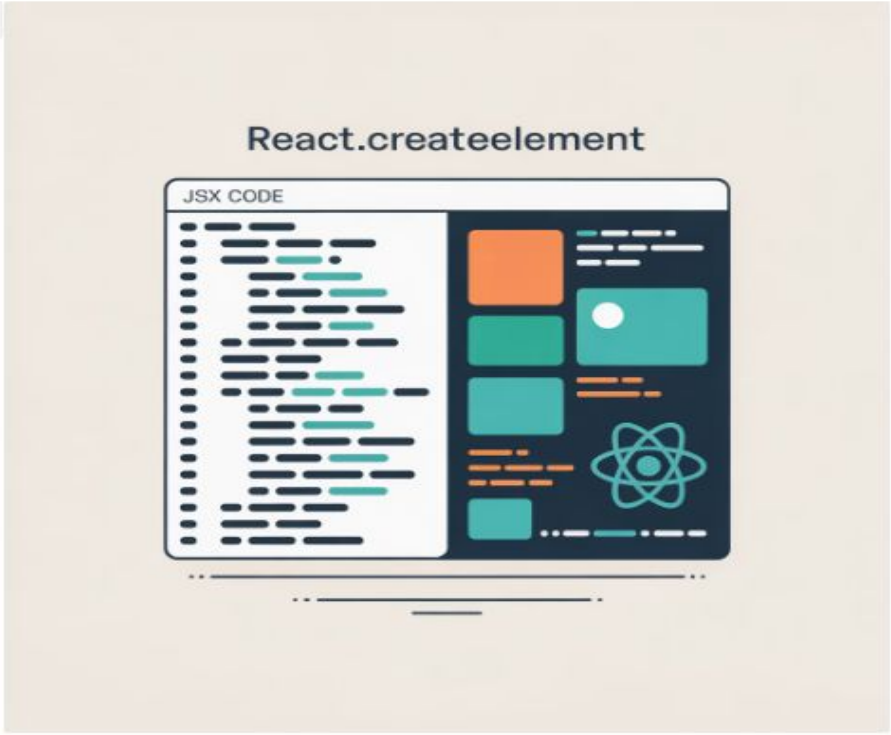
Ejercicio Práctico

Crear un componente "Tarjeta de Presentación" usando props.

¿Qué es JSX?

JSX es una extensión de sintaxis para JavaScript que **parece HTML pero no lo es**. Permite escribir código similar a HTML directamente dentro de JavaScript.

Bajo el capó, JSX se compila a llamadas a `React.createElement()`, generando elementos de React que representan nodos del DOM virtual.



JSX facilita la escritura de interfaces de usuario al combinar la potencia de JavaScript con la familiaridad de la sintaxis HTML.

JSX vs HTML: Diferencias Clave

1

camelCase

JSX usa camelCase para props y eventos:

```
// HTML: onclick, tabindex
// JSX: onClick, tabIndex
```

2

Atributos

Algunos nombres cambian para evitar palabras reservadas:

```
// HTML: class, for
// JSX: className, htmlFor
```

3

Expresiones JS

Integra JavaScript directamente usando llaves:

```
// JSX con expresiones
```

{user.name}

...

4

Elemento Padre

JSX requiere un único elemento padre:

```
// Incorrecto:
```

Título

Texto

// Correcto:

Título

Texto

Seamlessly visualize and prototype yc UI components with real-time code updates

¿Por qué usar JSX?

Legibilidad y expresividad

El código JSX es más intuitivo y fácil de entender que las llamadas a `React.createElement()`. Refleja claramente la estructura del árbol DOM que se generará.

Integración de lógica y markup

Permite combinar JavaScript y HTML de manera fluida, facilitando la creación de interfaces dinámicas que responden a datos y estados.

Desarrollo más eficiente

Acelera la creación de prototipos y reduce errores gracias a su sintaxis declarativa y al feedback inmediato del compilador.

Componentes: El Corazón de React

Los componentes son piezas de UI reutilizables que encapsulan tanto su apariencia como su comportamiento. Son los bloques de construcción fundamentales de cualquier aplicación React.

Componentes Funcionales

Funciones de JavaScript que reciben props y retornan elementos React (JSX).

```
function Welcome(props) {
  return
```

Hola, {props.name}

;}

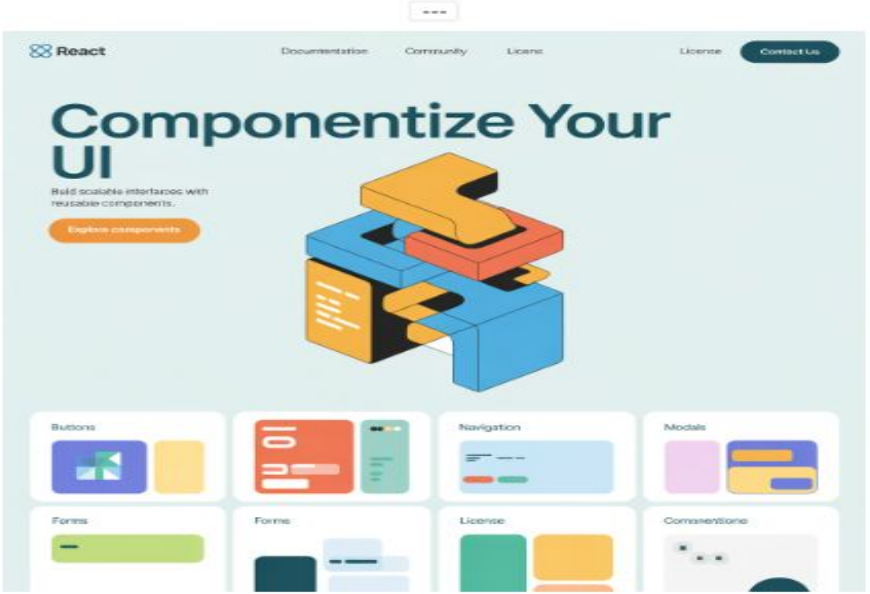
Componentes de Clase

Clases de ES6 que extienden React.Component y definen un método render().

```
class Welcome extends React.Component {
  render() {
    return
```

Hola, {this.props.name}

; }}



Cómo Funcionan los Componentes

Reutilizables

Pueden usarse múltiples veces en distintas partes de la aplicación, siguiendo el principio DRY (Don't Repeat Yourself).

Encapsulados

Cada componente maneja su propia lógica, estado y UI, lo que facilita el desarrollo y mantenimiento.

Independientes

Los cambios en un componente no afectan a otros, lo que facilita el aislamiento de problemas.

Componibles

Se pueden combinar para crear componentes más complejos, formando una jerarquía de componentes.

Este enfoque modular permite crear aplicaciones complejas a partir de piezas simples y reutilizables.

Funcionales vs. Clase

Componentes Funcionales

Funciones de JavaScript simples que aceptan "props" como argumento y retornan elementos React. Son más concisos y fáciles de leer.

- Usan **Hooks** para manejar el estado y los efectos secundarios (ej: `useState`, `useEffect`).
- Son el estándar moderno y preferido para la mayoría de los casos de uso en React.

Componentes de Clase

Clases de ES6 que extienden `React.Component` y deben tener un método `render()` que retorna elementos React.

- Manejan el estado interno con `this.state` y `this.setState()`.
- Utilizan **métodos de ciclo de vida** (ej: `componentDidMount`) para efectos secundarios.
- Aunque aún soportados, son menos utilizados en el desarrollo moderno de React.

Props: Enviando Información a los Componentes

Las props (abreviatura de "properties" o propiedades) son el mecanismo principal para pasar datos de un componente padre a un componente hijo.

```
// Componente padre
function App() {
  return ;
}

// Componente hijo
function UserCard(props) {
  return (
```

{props.name}

{props.role}

```
  ); }
```

1

Unidireccionales

Los datos fluyen de padres a hijos, nunca al revés.

2

Inmutables

Son de solo lectura dentro del componente que las recibe.

3

Flexibles

Pueden ser cualquier tipo de dato: strings, números, objetos, funciones, etc.

Ejemplo: Creando y Usando un Componente

```
// Definición del componente
function UserCard(props) {
  return (
```

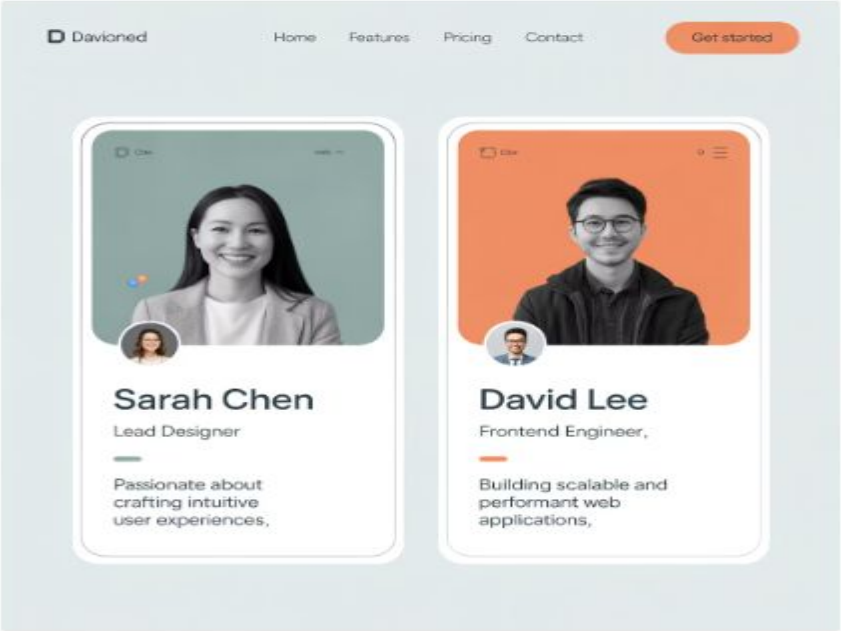
{props.name}

{props.role}

{props.bio}

); } // Uso del componente function App() { return (

); }



Los componentes permiten crear interfaces consistentes que se actualizan dinámicamente según los datos recibidos a través de props.

🚩 Ejercicio: Armá un Componente "Tarjeta de Presentación"

Objetivos:

1. Crear un componente `PresentationCard` que reciba props
2. Mostrar nombre, rol, foto y redes sociales usando los datos recibidos
3. Personalizar el estilo para que parezca una tarjeta de presentación
4. Probar el componente con diferentes datos para ver cómo se adapta

Estructura sugerida:

```
function PresentationCard(props) {
  return (

<h1>{props.name}</h1>

<p>{props.role}</p>

<p>{props.contact}</p>

  ); }
```

Uso del componente:

¿Alguna duda?

streambe GENERACIÓN T

Muchas gracias.

streambe GENERACIÓN T

Nuestras Redes

www.generaciont.org
www.streambe.com
www.instagram.com/generaciont_ar
www.tiktok.com/@generaciont
generaciont@generaciont.org
Cel: 11 61331747

GENERACIÓN T

streambe