BOOTCAMP

Generación T

streambe

Listas, Keys y Renderizado Dinámico en React

Una guía práctica para estudiantes de desarrollo web que están aprendiendo a trabajar con datos dinámicos en aplicaciones React.

Objetivos de Aprendizaje

01

Mostrar listas en React

Aprenderás a utilizar el método .map() para transformar arrays de datos en elementos JSX que React puede renderizar en el DOM.

02

Entender las keys

Comprenderás por qué React necesita identificadores únicos y cómo implementarlos correctamente para optimizar el rendimiento.

03

Renderizado condicional

Dominarás técnicas para mostrar elementos de forma dinámica según el estado de tu aplicación y los datos disponibles.

04

Manipular listas

Practicarás cómo agregar, eliminar y actualizar elementos en listas utilizando el hook useState para una experiencia interactiva.

Renderizando Listas con .map()

¿Qué es .map()?

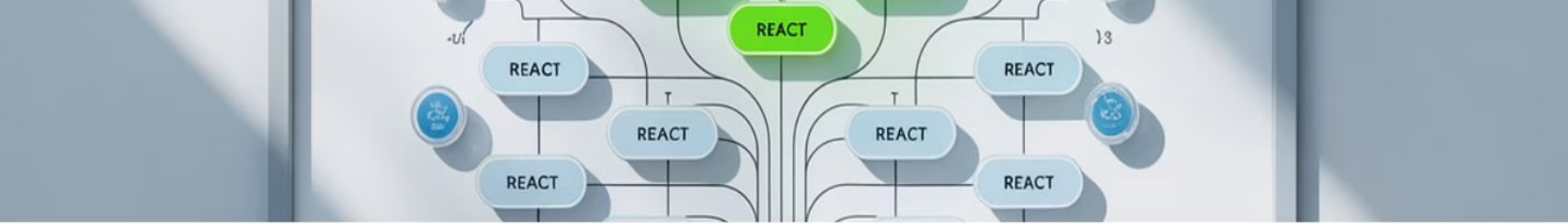
El método .map() es una función de los arrays en JavaScript que nos permite:

- Transformar cada elemento de un array
- Crear un nuevo array con los resultados
- Convertir datos en elementos JSX

En React, es la forma estándar de generar múltiples componentes similares a partir de una colección de datos.

```
// Ejemplo básico de .map() en React
function ListaFrutas() {
  const frutas = ['Manzana', 'Banana', 'Naranja'];

  return (


    {frutas.map(fruta => (           ))}
  • {fruta}
  ); }
```

Observa que cada elemento tiene un atributo `key` con un valor único. Esto es fundamental y lo explicaremos en detalle a continuación.



La Importancia de las Keys en React

¿Qué son las keys?

Son identificadores únicos que React utiliza para rastrear qué elementos se han actualizado, agregado o eliminado en una lista.

¿Por qué son necesarias?

Sin keys, React tendría que actualizar todo el DOM cuando cambia un elemento. Con keys, React identifica exactamente qué cambió, optimizando el rendimiento.

Buenas prácticas

- Usa IDs estables de tus datos
- Evita usar índices del array como keys (salvo excepciones)
- Las keys deben ser únicas entre hermanos, no globalmente

Un error común es usar `index` como key cuando el orden de los elementos puede cambiar, causando comportamientos inesperados en la interfaz.

Error Común: Índices como Keys

Problema con los índices

Cuando usamos el índice de array como key y el orden de los elementos cambia, React puede confundirse sobre qué elementos debe actualizar.

> Si usas índices como keys y reordenas, agregas o eliminas elementos, podrías tener problemas de rendimiento y bugs difíciles de detectar.

```
// Incorrecto (en listas dinámicas)
{tareas.map((tarea, index) => (

))}

// Correcto
{tareas.map(tarea => (

))}
```

Solo usa índices cuando:

- La lista nunca cambiará de orden
- No se agregarán/eliminarán elementos
- Los elementos no tienen IDs estables

Renderizado Condicional en React

El renderizado condicional te permite mostrar diferentes elementos según ciertas condiciones, haciendo tus interfaces más dinámicas e interactivas.

Operador ternario

```
{isLoggedIn
  ?
  : }
```

Operador &&

```
{tareas.length > 0 &&
  }
```

If/else en función

```
{function getContent() {
  if (isLoading) return ;
  if (error) return ;
  return ;
}}
```

Estos patrones son esenciales cuando trabajas con listas, ya que te permiten mostrar mensajes especiales para listas vacías o visualizaciones alternativas según los datos disponibles.

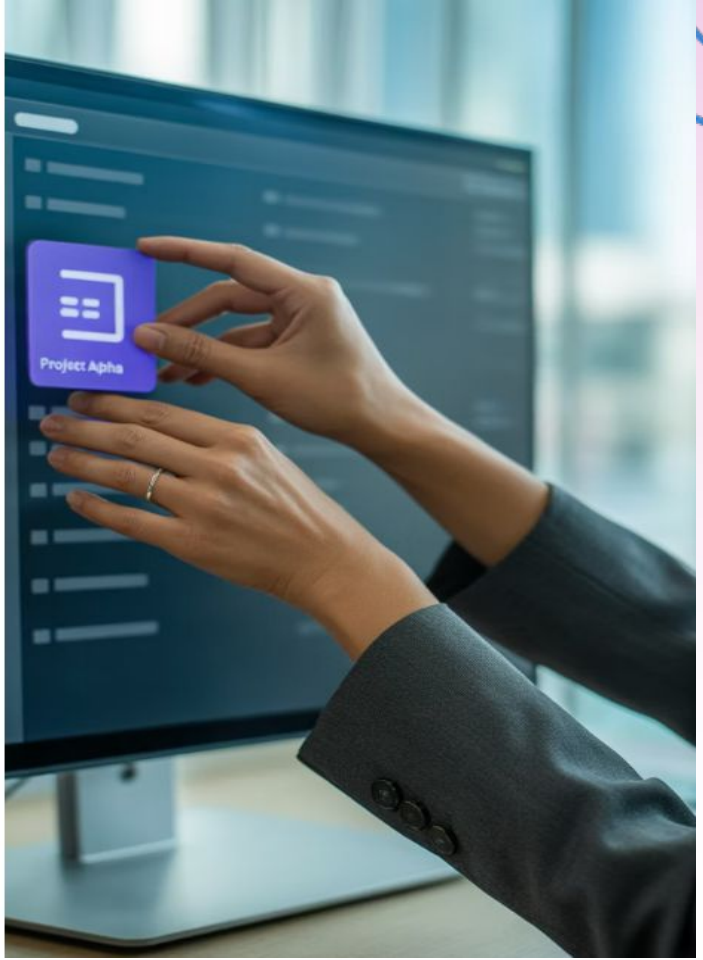
Manipulación de Listas con useState

Principios importantes

- **Inmutabilidad**: Nunca modifiques el state directamente
- **Nuevas referencias**: Crea nuevos arrays para actualizar
- **Métodos puros**: Usa map, filter, spread, concat

React compara referencias de objetos, no contenido. Si modificas el array original, React podría no detectar el cambio y no volver a renderizar.

```
// Agregar un elemento
setTareas([...tareas,
nuevaTarea]);

// Eliminar un elemento
setTareas(
 tareas.filter(tarea =>
tarea.id !== idAEliminar)
);

// Actualizar un elemento
setTareas(
 tareas.map(tarea =>
 tarea.id === idAActualizar
 ? { ...tarea, completada:
true }
 : tarea
 )
);
```



Patrones Avanzados de Renderizado

Renderizado por componentes

Extrae la lógica de renderizado a componentes especializados que se ocupan de diferentes estados:

```
function EstadoLista({ tareas,
isLoading, error }) {
  if (isLoading) return ;
  if (error) return ;
  if (tareas.length === 0)
return ;
  return ;
}
```

Componentes de orden superior (HOC)

Utiliza HOCs para añadir funcionalidades a múltiples listas:

```
function
conManejoDeCarga(Component) {
  return function ({
isLoading, ...props }) {
    if (isLoading) return ;
    return ;
  }
}

const ListaTareasConCarga =
conManejoDeCarga(ListaTareas);
```

Render props

Permite a un componente compartir lógica a través de una prop de función:

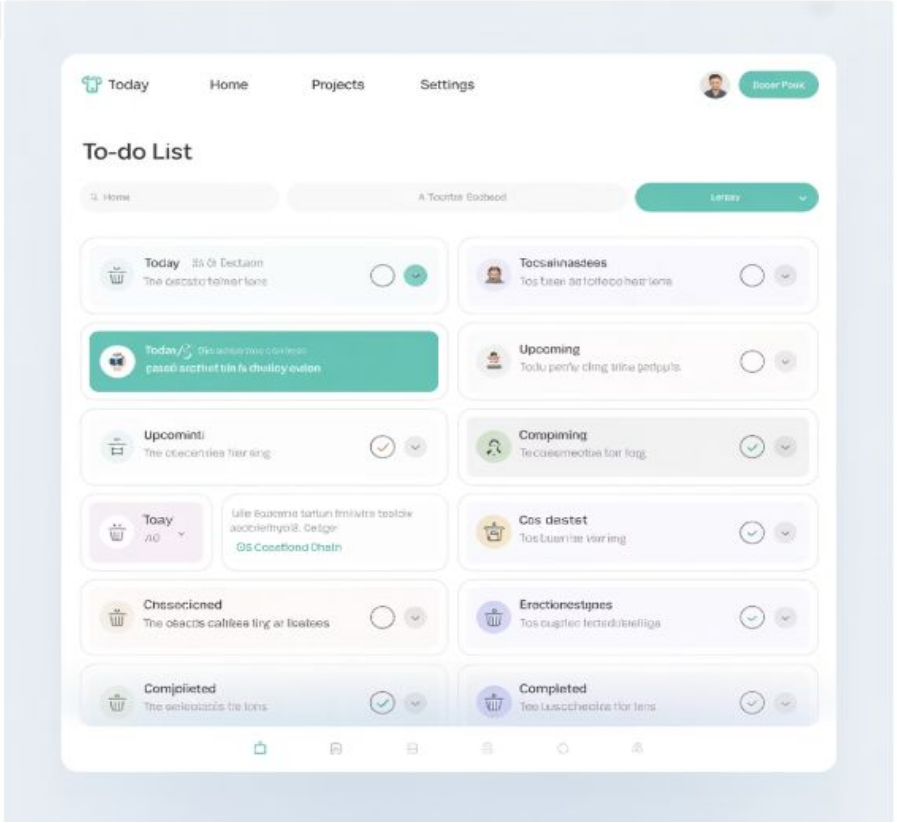
```
  (

  )}
/>
```

Estos patrones te ayudarán a manejar listas más complejas y a reutilizar lógica entre diferentes componentes que trabajan con colecciones de datos.

Ejercicio Práctico

Crea una aplicación de lista de tareas con estas características:

- Mostrar tareas desde un array con .map()
- Implementar un formulario para agregar nuevas tareas
- Agregar botones para eliminar tareas específicas
- Incluir función para marcar tareas como completadas
- Mostrar mensaje especial cuando la lista esté vacía
- Asegurar que usas keys apropiadas en todos los elementos de la lista



¿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