React

Patrones comunes

Enrique Barra Arias
Álvaro Alonso González

Renderizando desde una lista

- En React, es común generar un grupo de componentes a partir de un array de datos
- Para ello, se suele utilizar la función **map**

```
let cooked = [🐮, 🍆, 🐔, 🌽, 🐟, 🐖, 🍋, 🍓]
    .map(cook) // [🍔, 🍟, 🍗, 🍿, 🌭, 🍸, 🍨]
```

Ejemplo de array

- https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/ListaProductos.js
- https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/Producto.js
- Si lo queremos visualizar ponemos en index.js lo siguiente:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import ListaProductos from './ListaProductos';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <ListaProductos />
  </React.StrictMode>
);
```

Componente reusable

```
//componente producto, que recibe como props
//productName y costInEuros y pinta un producto
export default function Producto(props){
  return <div className="my-product">
    <b>{props.productName}</b>
    <span className="right-aligned">
      ({props.costInEuros} €)
    </span>
    </div>
}
```

Generar lista de componentes

Tenemos que iterar a través de la lista usando la función map y crear para cada elemento de la lista una instancia del componente Producto, pasándole las props con el nombre correcto

```
import Producto from "./Producto";

export default function ListaProductos(props) {
  const  productList  = [
    { name: "Coca-cola", price: 0.45 },
    {name: "Chocolate",price: 1.24},
    {name: "Popcorn", price: 1.98}
  ];

  return (<div>
      <h1>Shopping cart</h1>
      { productList.map((product, index) => {
          const { name, price } = product;
          return <Producto key={index}
              productName={name}
              costInEuros={price} />
        })
      }
  </div>)
}
```

Bajar datos del servidor

- La obtención de datos de un servidor se hace normalmente una vez que nuestra aplicación se carga
- Por lo tanto, debe realizarse en el hook useEffect
- Una vez que los datos han sido descargados del servidor, podemos actualizar el estado de nuestro componente
- NOTA: si los datos se descargan al pulsar un botón eso se hará en la función que llame dicho botón
- https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/CargaDatos.js

Bajar datos del servidor

```
export default function CargaDatos(props){
  const url = 'https://dummyjson.com/users/1';
  const [data, setData] = useState(null);

  useEffect(() => {
    async function fetchData() {
      try {
        const response = await fetch(url);
        if (response.ok) {
          const data = await response.json();
          setData(data);
        } else {
          console.log('Respuesta de red OK pero respuesta de HTTP no OK');
        }
      } catch(e) {
        console.log("ERROR", e);
      }
    }

    fetchData();
  }, []);

  return <div>
    <b>Datos cargados de la URL: {url}</b>
    <pre>
      {JSON.stringify(data, null, 2)}
    </pre>
    </div>
}
```

Tratamiento de errores: componentDidCatch

- **componentDidCatch**(error, info)
- Funciona como un bloque try catch en JavaScript pero para componentes
- Es un método especial de ciclo de vida que transformará el componente **de clase** en un error boundary
- Los error boundaries capturan errores durante el renderizado de todo el árbol de componentes bajo ellos
- Los error boundaries no capturan los errores en los manejadores de eventos o en código asíncrono (se debe usar try/catch en esos casos)
- Llamar a setState() en un error boundary permite capturar un error en el código y mostrar una interfaz de fallback en base al error

- Más info: https://reactjs.org/docs/error-boundaries.html

Tratamiento de errores: componentDidCatch

La mayoría de las veces querrá declarar un componente de error boundary una vez y utilizarlo en toda su aplicación.

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }
  componentDidCatch(error, info) {
    // Display fallback UI
    this.setState({ hasError: true });
    logErrorToMyService(error, info);
  }
  render() {
    if (this.state.hasError) {
      return <h1>Something went wrong.</h1>;
    }
    return this.props.children;
  }
}
```

```
<ErrorBoundary>
  <MyWidget />
</ErrorBoundary>
```