«Talento Tech»

Desarrollo Web 3

Clase 08

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 8 | Props vs. Context API

Temario:

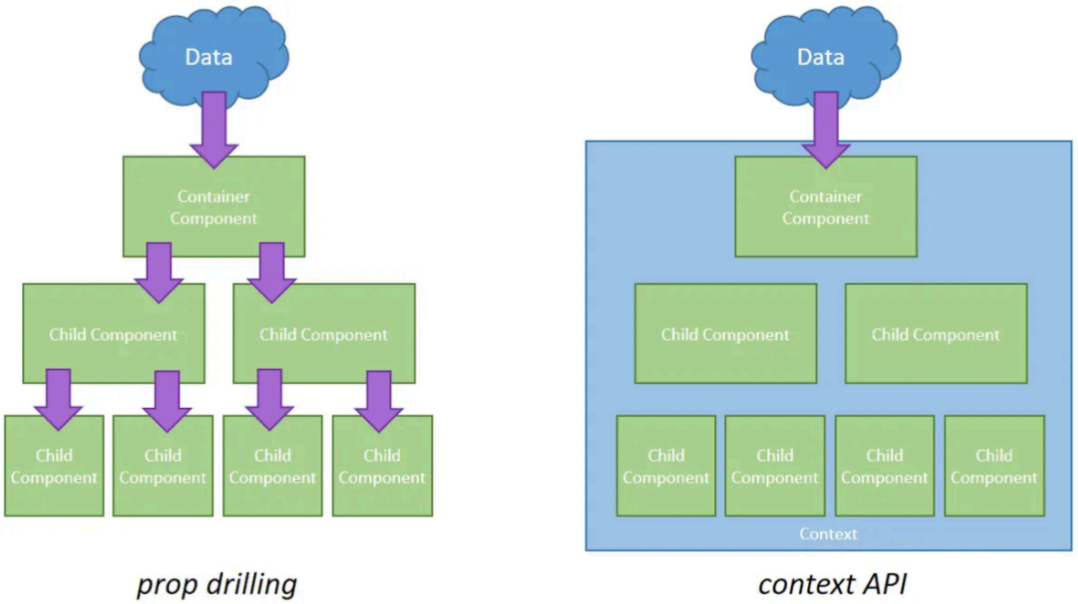
- Introducción a Context Api
- Creación de un Contexto
- Proveer y consumir un Context

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Introducción a Context Api

Hasta el momento habíamos visto una forma para poder comunicar componentes de padre a hijo a través de las props. Ahora vamos a aprender una nueva forma de generar esta comunicación de forma más eficiente y ordenada que será a través de **Context API.** Para poder adentrarnos mejor en el tema vamos a primero conocer qué es **Context**, cómo funciona y para qué se usa.

Context API es una característica de **React** que proporciona una forma de pasar datos a través del árbol de componentes sin necesidad de pasar explícitamente las props a través de cada nivel del árbol. Permite compartir valores como el estado, la configuración o cualquier dato que se considere global o compartido por muchos componentes.



Context API consta de dos partes principales

1. **createContext** : Una función que crea un nuevo contexto. Puedes proporcionar un valor predeterminado como argumento, que se utilizará cuando un componente no esté envuelto en un Provider correspondiente.

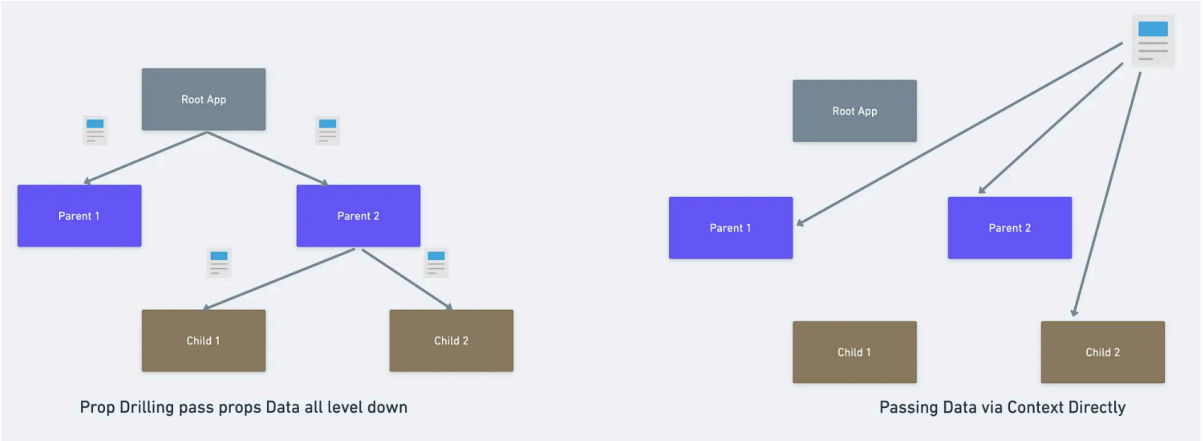
2. **<Provider>** y **useContext** : Componentes que se utilizan para proporcionar y consumir el contexto.

- <Provider> : Este componente se utiliza para envolver parte o toda la aplicación y proporcionar el valor del contexto a los componentes descendientes.

- useContext : Es un hook de React que permite a los componentes acceder al valor de un contexto proporcionado por el **Context.Provider** más cercano en la jerarquía de componentes, sin necesidad de pasar propiedades manualmente a través de los componentes intermedios.

La Context API es especialmente útil en situaciones donde múltiples componentes necesitan acceder a un mismo estado o datos globales, evitando así la **prop drilling (pasar props a través de múltiples niveles de componentes).** Es importante tener en cuenta que, aunque la Context API es poderosa, no siempre es la mejor opción para todos los casos, y su uso debe evaluarse considerando la complejidad y estructura de la aplicación.

Context APi se suele utilizar para casos como manejar el estado global de autenticación, temas de la aplicación, preferencias del usuario o cualquier dato compartido por muchos componentes.



Buenos Aires aprende
Agencia de Habilidades para el Futuro

Creación de un Context

A continuación vamos a ver los primeros pasos para poder crear un contexto. En este caso vamos a hacer un mini proyecto de “alta baja” y modificación de tareas el cual nos permitirá comprender un poco mejor cómo funciona éste nuevo sistema.

Primero lo que debemos hacer en nuestro proyecto de React es crear una carpeta que se llame **“context”** la cual vamos a crear un archivo que se llame **“Tareas.js”.** ¿Por qué hacemos esto? Es muy simple. Es importante que nuestro proyecto esté separado por carpetas para poder tener una mejor organización y a su vez poder separar en el caso que necesitemos más de un contexto. Esto se debe a que, por lo general, solemos usar más de un contexto en nuestra aplicación.

Por ejemplo, si estuviéramos trabajando en un proyecto en el cual se requiera un escenario de autenticación (login, registro), un carrito de compras, etc. podríamos usar un contexto para cada caso de uso. Esto nos permitirá tener nuestro proyecto segmentado y nos brindará la posibilidad de escalar en nuevas features.

Luego vamos a necesitar importar **createContext** y **useState** en nuestro archivo **Tareas.js** de la siguiente manera:

```
import  { createContext, useState } from 'react';
```

A continuación, almacenaremos en una variable el contexto creado de la siguiente forma.

```
const Tareas = createContext();
```

Proveer y consumir un Context

Creación del Context
Debajo vamos a crear el Componente el cual será el proveedor de nuestra aplicación.

```
const ProveedorDeTareas = ({children}) => {
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

```
const tareasIniciales = [
   { id: 1, nombre: 'Hacer la compra', descripcion: 'Comprar alimentos y productos
necesarios.' },
   { id: 2, nombre: 'Estudiar React', descripcion: 'Revisar la documentación y hacer
ejercicios.' },
   { id: 3, nombre: 'Ir al gimnasio', descripcion: 'Realizar ejercicios de cardio y pesas.' },
 ];

 const [tareas, setTareas] = useState(tareasIniciales);

 return (
     <Tareas.Provider value={tareas}>
      {children}
   </Tareas.Provider>
 );
};

export default ProveedorDeTareas;
```

En este bloque de código podemos observar que por encima del return podemos agregar hooks como **useState**, el cual nos permitirá controlar un estado global que podrá ser mutado en cualquier parte de nuestra app, siempre y cuando se encuentre dentro del **scope** de nuestro **contexto global**.

```
const tareasIniciales = [
   { id: 1, nombre: 'Hacer la compra', descripcion: 'Comprar alimentos y productos necesarios.' }
   { id: 2, nomnbre: 'Estudiar React', descripcion: 'Revisar la documentación y hacer ejercicios' },
   { id: 3, nombre: 'Ir al gimnasio', descripcion: 'Realizar ejercicios de cardio y pesas' },
];

   const [tareas, setTareas] = useState(tarerasIniciales);
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Agregamos funcionalidades al Contexto

Ahora vamos a agregar algunas funcionalidades correspondientes al **ABM (Alta, Baja y Modificación)** que nos van a permitir mutar el estado de las tareas para poder aplicarlas luego en nuestra **UI (user interface).** Estas funcionalidades las ubicaremos debajo del estado de la siguiente forma:

```
const agregarTarea = (nuevaTarea) => {
    setTareas([...tareas, nuevaTarea]);
  };

  // Función para eliminar una tarea
  const eliminarTarea = (id) => {
    const nuevasTareas = tareas.filter((tarea) => tarea.id !== id);
    setTareas(nuevasTareas);
  };

  // Función para editar una tarea
  const editarTarea = (id, nuevaTarea) => {
    const nuevasTareas = tareas.map((tarea) =>
      tarea.id === id ? { ...tarea, ...nuevaTarea } : tarea
    );
    setTareas(nuevasTareas);
  };
```

Por debajo del return lo que estamos haciendo es generar el Proveedor el cual va a pasar por props la información que queramos compartir en el contexto a todos los componentes hijos (children).

```
return (
      <tareas.Provider value={tareas}>
      {children}
    </Tareas.Provider>
    );
};
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

En **React**, el ".Provider" es un componente que forma parte de la **API de Context**. Este componente se utiliza para envolver partes de tu aplicación y proporcionar el valor del contexto a los componentes descendientes. El .**Provider** acepta una prop llamada "value", que especifica el valor del contexto que debe estar disponible para los componentes dentro del árbol de ese Provider.

Nuestro contexto va a tener que quedar de la siguiente forma :

```
import { createContext, useState } from 'react';

const TareasContext = createContext();

const ProveedorDeTareas = ({ children }) => {
  const tareasIniciales = [
    { id: 1, nombre: 'Hacer la compra', descripcion: 'Comprar alimentos y productos
necesarios.' },
    { id: 2, nombre: 'Estudiar React', descripcion: 'Revisar la documentación y hacer
ejercicios.' },
    { id: 3, nombre: 'Ir al gimnasio', descripcion: 'Realizar ejercicios de cardio y pesas.' },
  ];

  const [tareas, setTareas] = useState(tareasIniciales);

  // Función para agregar una nueva tarea
  const agregarTarea = (nuevaTarea) => {
    setTareas([...tareas, nuevaTarea]);
  };

  // Función para eliminar una tarea
  const eliminarTarea = (id) => {
    const nuevasTareas = tareas.filter((tarea) => tarea.id !== id);
    setTareas(nuevasTareas);
  };

  // Función para editar una tarea
  const editarTarea = (id, nuevaTarea) => {
    const nuevasTareas = tareas.map((tarea) =>
```

```
      tarea.id === id ? { ...tarea, ...nuevaTarea } : tarea
    );
    setTareas(nuevasTareas);
  };

  return (
    <TareasContext.Provider value={{ tareas, agregarTarea, eliminarTarea, editarTarea }}>
      {children}
    </TareasContext.Provider>
  );
};

export { ProveedorDeTareas, TareasContext };
```

Importamos el Contexto en nuestro componente principal

Para poder consumir nuestro contexto en nuestro proyecto vamos a tener que ir a nuestro componente principal **App.js**, vamos a tener que importar el contexto creado y lo haremos utilizando la línea de código “import React from 'react'”. Podemos visualizarlo en la siguiente imagen:

```
import ProveedorDeTareas from './context/tareas.js';
const App = () => {
  return (
    <ProveedorDeTareas >
          // Aca van a ir el resto de los componentes que uses en tu aplicación
    </ProveedorDeTareas >
  );
};


export default App;
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

```
const proveedorDeTareas = ({children}) => {
   const tareasIniciales = [
      {id: 1, nombre: "Hacer la compra", descripcion: "Comprar alimentos y productos
necesarios"},
      {id: 2, nombre "Estudiar React", descripcion: "Revisar la documentacion y hacer ejercicios"},
      {id: 3, nombre: "Ir al gimnasio", descripcion: "Realizar ejercicios de cardio y pesas"},
   ];
   const [tareas, setTareas] = useState(tareasIniciales);
   return(
      <Tareas.Provider value={tareas}>
         {children}
      </Tareas.Provider>
   );
};
export default proveedorDeTareas;
```

🌟Bonus Extra (UI)

Desarrollamos la UI de nuestro contexto

Para poder comenzar vamos a necesitar tener instalado react-router-dom en nuestro proyecto para que podamos trabajar con vistas.

Para eso vamos a crear una carpeta llamada “Components” y dentro de ella vamos a crear una carpeta llamada “Navbar” donde crearemos un archivo “Navbar.js” para nuestro componente funcional y otro llamado “Navbar.css” para agregar los estilos.

Creación del Navbar

Vamos a agregar el siguiente componente: vamos a agregar el componente “Navbar.” de la siguiente manera:

Buenos Aires aprende
Agencia de Habilidades para el Futuro

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import './Navbar.css';

const Navbar = () => {
  const [menuAbierto, setMenuAbierto] = useState(false);

  const toggleMenu = () => {
    setMenuAbierto(!menuAbierto);
  };
  return (
    <nav>
      <div className={`menu ${menuAbierto ? 'abierto' : ''}`}>
        <div className="menu-icon" onClick={toggleMenu}>
          <div className="bar"></div>
          <div className="bar"></div>
          <div className="bar"></div>
        </div>
        <ul>
          <li>
            <Link to="/agregar" onClick={toggleMenu}>
              Agregar Tarea
            </Link>
          </li>
          <li>
            <Link to="/ver-todas" onClick={toggleMenu}>
              Ver Todas las Tareas
            </Link>
          </li>
          <li>
            <Link to="/editar/1" onClick={toggleMenu}>
              Editar Tarea
            </Link>
          </li>
        </ul>
      </div>
    </nav>
```

```
  );
};

export default Navbar;
```

Vamos a crear ahora nuestro “CSS”:

```
nav {
  background-color: #333;
  padding: 10px;
}

.menu {
  display: flex;
  align-items: center;
}

.menu ul {
  list-style-type: none;
  padding: 0;
  margin: 0;
  display: flex;
}

.menu li {
  margin-right: 20px;
}

.menu a {
  text-decoration: none;
  color: white;
}

.menu-icon {
  cursor: pointer;
  display: none;
```

```
  flex-direction: column;
}

.bar {
  background-color: white;
  height: 5px;
  width: 30px;
  margin: 6px 0;
}

.abierto .menu ul {
  display: flex;
  flex-direction: column;
  position: absolute;
  top: 60px;
  left: 0;
  background-color: #333;
  width: 100%;
  z-index: 1;
}

.abierto .menu-icon {
  display: flex;
}
```

Incorporación de rutas a App.js

Define rutas para agregar, editar y ver tareas, y utiliza el componente ProveedorDeTareas para gestionar el estado relacionado con las tareas. La navegación se logra con un componente Navbar y el uso de **<Switch> y <Route>** de **React Router.**

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import { ProveedorDeTareas } from './ProveedorDeTareas';
import AgregarTarea from './AgregarTarea';
```

```
import EditarTarea from './EditarTarea';
import VerTodasLasTareas from './VerTodasLasTareas';
import Navbar from './Navbar';

const App = () => {
  return (
    <Router>
      <ProveedorDeTareas>
        <Navbar />
        <Switch>
          <Route path="/agregar" component={AgregarTarea} />
          <Route path="/editar/:id" component={EditarTarea} />
          <Route path="/ver-todas" component={VerTodasLasTareas} />
          {/* Otras rutas si es necesario */}
        </Switch>
      </ProveedorDeTareas>
    </Router>
  );
};

export default App;
```

Componente “Agregar Tarea”

En la carpeta “components” creamos la carpeta “Agregar” y dentro de ella creamos el archivo “AgregarTarea.js” el cual contendrá toda la lógica del componente funcional.

```
import React, { useContext, useState } from 'react';
import { TareasContext } from './ProveedorDeTareas';

const AgregarTarea = () => {
  const { agregarTarea } = useContext(TareasContext);
  const [nuevaTarea, setNuevaTarea] = useState({ nombre: '', descripcion: '' });

  const handleInputChange = (e) => {
    const { name, value } = e.target;
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

```
    setNuevaTarea((prevTarea) => ({
      ...prevTarea,
      [name]: value,
    }));
  };

  const handleAgregarTarea = () => {
    // Validar y agregar la nueva tarea
    if (nuevaTarea.nombre.trim() === '' || nuevaTarea.descripcion.trim() === '') {
      alert('Por favor, completa todos los campos.');
      return;
    }

    agregarTarea({
      id: Date.now(),
      ...nuevaTarea,
    });

    // Limpiar los campos después de agregar la tarea
    setNuevaTarea({ nombre: '', descripcion: '' });
  };

  return (
    <div>
      <h2>Agregar Tarea</h2>
      {/* Formulario para agregar tarea */}
      <form>
        <label>
          Nombre:
          <input type="text" name="nombre" value={nuevaTarea.nombre}
onChange={handleInputChange} />
        </label>
        <br />
        <label>
          Descripción:
          <textarea
            name="descripcion"
```

```
          value={nuevaTarea.descripcion}
          onChange={handleInputChange}
        ></textarea>
      </label>
      <br />
      <button type="button" onClick={handleAgregarTarea}>
        Agregar Tarea
      </button>
    </form>
  </div>
 );
};

export default AgregarTarea;
```

Componente “editar Tarea”

En la carpeta “components” creamos la carpeta “Editar” y dentro de ella creamos el archivo “EditarTarea.js” el cual contendrá toda la lógica del componente funcional.

```
import React, { useContext, useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';
import { TareasContext } from './ProveedorDeTareas';

const EditarTarea = () => {
  const { id } = useParams();
  const { tareas, editarTarea } = useContext(TareasContext);
  const [tareaEditada, setTareaEditada] = useState({ nombre: '', descripcion: '' });

  useEffect(() => {
    // Obtener la tarea específica por ID y asignarla a tareaEditada
    const tareaSeleccionada = tareas.find((tarea) => tarea.id === parseInt(id, 10));
    if (tareaSeleccionada) {
      setTareaEditada(tareaSeleccionada);
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

```
  }
}, [id, tareas]);

const handleInputChange = (e) => {
  const { name, value } = e.target;
  setTareaEditada((prevTarea) => ({
    ...prevTarea,
    [name]: value,
  }));
};

const handleEditarTarea = () => {
  // Validar y editar la tarea
  if (tareaEditada.nombre.trim() === '' || tareaEditada.descripcion.trim() === '') {
    alert('Por favor, completa todos los campos.');
    return;
  }

  editarTarea(parseInt(id, 10), tareaEditada);
  // Omitir la redirección
};

return (
  <div>
    <h2>Editar Tarea</h2>
    {/* Formulario para editar tarea */}
    <form>
      <label>
        Nombre:
        <input
          type="text"
          name="nombre"
          value={tareaEditada.nombre}
          onChange={handleInputChange}
        />
      </label>
      <br />
```

```
        <label>
          Descripción:
          <textarea
            name="descripcion"
            value={tareaEditada.descripcion}
            onChange={handleInputChange}
          />
        </label>
        <br />
        <button type="button" onClick={handleEditarTarea}>
          Guardar Cambios
        </button>
      </form>
    </div>
  );
};

export default EditarTarea;
```

Componente “Ver Todas las Tareas”

En la carpeta “components” creamos la carpeta “Editar”y dentro de ella creamos el archivo “VerTodasLasTareas.js” el cual contendrá toda la lógica del componente funcional.

```
import React, { useContext, useState, useEffect } from 'react';
import { TareasContext } from './ProveedorDeTareas';
import { Link } from 'react-router-dom';

const VerTodasLasTareas = () => {
  const { tareas, eliminarTarea } = useContext(TareasContext);
  const [tareasSeleccionadas, setTareasSeleccionadas] = useState([]);
  const [tareaSeleccionada, setTareaSeleccionada] = useState(null);

  useEffect(() => {
      setTareasSeleccionadas(tareas.map(tarea => tarea.id));
```

```
  }, [tareas]);

  const handleEliminarTarea = (id) => {
    eliminarTarea(id);
     };

  const handleSeleccionarTarea = (id) => {
    setTareaSeleccionada(id);
  };

  return (
    <div>
      <h2>Ver Todas las Tareas</h2>
      {tareas.length === 0 ? (
        <p>Cargando...</p>
      ) : (
        <div>
          <ul>
            {tareas.map((tarea) => (
              <li key={tarea.id}>
                {tarea.nombre} - {tarea.descripcion}
                <button onClick={() => handleEliminarTarea(tarea.id)}>Eliminar</button>
                <button onClick={() => handleSeleccionarTarea(tarea.id)}>Seleccionar</button>
              </li>
            ))}
          </ul>
          {tareasSeleccionadas.length > 0 && tareaSeleccionada && (
            <Link to={`/editar/${tareaSeleccionada}`}>Editar Tarea Seleccionada</Link>
          )}

        </div>
      )}
    </div>
  );
};

export default VerTodasLasTareas;
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Con esto ya quedaría nuestro proyecto terminado 🏆🎉


Buenos Aires
aprende
Buenos
Aires
Ciudad