«Talento Tech»

React JS

Clase 09

Buenos Aires Ciudad

Clase N° 9 | Autenticación de usuarios

Índice:

Objetivos de la Clase:

- Context API y cómo se puede usar para manejar el estado de la autenticación.
- Implementar un formulario de login básico que permita autenticar a un usuario.
- Manejar la autenticación mediante tokens simulados para prácticas de seguridad.
- Proteger rutas utilizando Context API para manejar el estado de autenticación y permitir que solo los usuarios autenticados accedan a ciertas secciones de la aplicación.

Introducción a la Autenticación de Usuarios

¿Qué es la autenticación?

La autenticación es el proceso mediante el cual una aplicación verifica la identidad de un usuario. Generalmente, se realiza mediante un formulario de login donde el usuario ingresa sus credenciales (como nombre de usuario y contraseña). Una vez autenticado, el usuario recibe un token de acceso que puede utilizar para realizar solicitudes protegidas.

¿Por qué es importante proteger rutas?

En aplicaciones web modernas, es común que algunas páginas o funcionalidades solo sean accesibles a usuarios autenticados, como perfiles de usuario, configuraciones personales, y carritos de compras. Para esto, se implementan "rutas protegidas", que verifican si un usuario está autenticado antes de permitirle acceder a esa página.

Implementación de Formulario de Login

Formulario de Login

Comenzamos creando un formulario básico de login en React que permita al usuario ingresar sus credenciales (nombre de usuario y contraseña). Para este ejemplo, vamos a simular la autenticación sin conectarnos a un backend real.

```jsx
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { useAuthContext } from '../context/AuthContext';

function Login() {
  const [usuario, setUsuario] = useState('');
  const [password, setPassword] = useState('');
  const { login } = useAuthContext();
  const navigate = useNavigate();

  const handleSubmit = (e) => {
    e.preventDefault();
    // Simulación de autenticación
    if (usuario === 'admin' && password === '1234') {
      login(usuario);
      navigate('/dashboard');
    } else {
      alert('Credenciales incorrectas');
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <h2>Iniciar sesión</h2>
      <div>
        <label>Usuario:</label>
        <input
          type="text"
          value={usuario}
          onChange={(e) => setUsuario(e.target.value)}
        />
      </div>
      <div>
        <label>Contraseña:</label>
        <input
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
      </div>
      <button type="submit">Iniciar sesión</button>
    </form>
```

```
  );
}

export default Login;
```

- En este formulario, estamos usando `useState` para manejar los valores de usuario y contraseña.
- La función `handleSubmit` simula la autenticación verificando si las credenciales ingresadas son correctas (usuario "admin" y contraseña "1234").
- Si las credenciales son correctas, usamos `login` para cambiar el estado global de autenticación y redirigir al usuario a la página de dashboard.

Manejo de Autenticación con Tokens (Simulada)

Simulación de Tokens

Los tokens generalmente se utilizan en aplicaciones reales para autenticar solicitudes a un servidor. En esta clase, vamos a simular el proceso de manejo de tokens usando Context API.

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

// Crear el contexto de autenticación

const AuthContext = createContext();

export function AuthProvider({ children }) {

  const [user, setUser] = useState(null);
```

```
  const login = (username) => {
    // Simulando la creación de un token (en una app real, esto sería
generado por un servidor)
    const token = `fake-token-${username}`;
    localStorage.setItem('authToken', token);
    setUser(username);
  };
  const logout = () => {
    localStorage.removeItem('authToken');
    setUser(null);
  };
  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider> );
}
export const useAuthContext = () => useContext(AuthContext);
```

- En este contexto, login guarda un token simulado en localStorage para simular que el usuario está autenticado.
- logout elimina el token del almacenamiento local y limpia el estado de usuario.
- useAuthContext es un hook personalizado que permite acceder al estado de autenticación desde cualquier componente.

Protección de Rutas usando Context API

Protección de Rutas

Las rutas protegidas solo deben ser accesibles si el usuario está autenticado. Para esto, usamos un componente de "protección de ruta" que verifica si el usuario está autenticado antes de renderizar la ruta deseada.

```
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useAuthContext } from '../context/AuthContext';

function ProtectedRoute({ children }) {
  const { user } = useAuthContext();

  if (!user) {
    return <Navigate to="/login" />;
  }

  return children;
}

export default ProtectedRoute;
```

- ProtectedRoute es un componente que actúa como una "valla de seguridad" para las rutas protegidas.
- Si no hay un usuario autenticado (!user), el componente redirige a la página de login usando Navigate.
- Si el usuario está autenticado, renderiza los componentes hijos (la ruta protegida).

Integración en la Aplicación

Componente Principal con Rutas Protegidas

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from
'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import Login from './components/Login';
import Dashboard from './components/Dashboard';
import ProtectedRoute from './components/ProtectedRoute';

function App() {
  return (
    <AuthProvider>
      <Router>
        <Routes>
          <Route path="/login" element={<Login />} />
          <Route
            path="/dashboard"
            element={
              <ProtectedRoute>
                <Dashboard />
              </ProtectedRoute>
            }
          />
        </Routes>
      </Router>
    </AuthProvider>
  );
}

export default App;
```

- Usamos ProtectedRoute para proteger la ruta /dashboard.
- Si el usuario no está autenticado, se redirige a la página de login

Nuevo objetivo en Talento Lab

En esta etapa del proyecto, el cliente nos ha pedido dos grandes mejoras:

1. Mejorar la experiencia de compra implementando un carrito de compras con estado global.
2. Asegurarnos de que solo usuarios autenticados puedan acceder a ciertas secciones del sitio.

Objetivo:

Crear código que integre el manejo del estado global para el carrito de compras y la autenticación de usuarios utilizando Context API y rutas protegidas con React Router DOM.

Parte 1: Gestión del Carrito de Compras

1. **Crear un Contexto para el Carrito:**
   - Crear un nuevo archivo `CarritoContext.js` en la carpeta `context`.
   - Implementar un contexto con las funciones para agregar productos al carrito y vaciarlo.
2. **Integrar el Contexto en la Aplicación:**
   - Envolver el componente principal (`App.js`) con el proveedor del contexto `CarritoProvider`.

Parte 2: Autenticación de Usuarios

1. **Crear un Contexto para la Autenticación**:
    - Crear un archivo `AuthContext.js` en la carpeta `context`.
    - Implementar funciones para iniciar sesión (`login`) y cerrar sesión (`logout`) simulando el manejo de un token con `localStorage`.
2. **Formulario de Login:**
    - Crear un componente `Login` con un formulario básico para que los usuarios ingresen un nombre de usuario y contraseña.
    - Simular la autenticación validando las credenciales y redirigir a la página principal si son correctas.
3. **Rutas Protegidas:**
    - Crear un componente `ProtectedRoute` que permita acceder a ciertas páginas solo si el usuario está autenticado.
    - Proteger la ruta del carrito para que solo usuarios autenticados puedan verlo.

Reflexión final

Aprendimos a gestionar la autenticación de usuarios en una aplicación React utilizando Context API. Creamos un formulario de login simulado, manejamos tokens de manera local, y protegimos rutas para garantizar que solo los usuarios autenticados pudieran acceder a ellas.

El uso de Context API nos permitió gestionar el estado global de autenticación de manera sencilla y eficiente, sin necesidad de "prop drilling". Esta es una solución escalable y moderna para gestionar el estado de la aplicación en React.

Materiales y Recursos Adicionales:

★ Documentación oficial de Context API

★ Documentación oficial de React Router

Preguntas para Reflexionar:

- ¿Por qué es más seguro manejar la autenticación con tokens en vez de mantener el estado de usuario en el componente?
- ¿Cómo manejarías la expiración de los tokens y la reautenticación?
- ¿Qué consideraciones de seguridad deben tenerse en cuenta al usar tokens en un entorno de producción?

Próximos Pasos:

- Creación de formulario para agregar productos.
- Validación de datos de formularios.
- Implementación de la funcionalidad para agregar productos.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad