



Agencia de Habilidades  
para el Futuro

<Talento  
Tech />

# React JS

---

**Clase 09 | Autenticación de usuarios**

# ¡Les damos la bienvenida!



Vamos a comenzar a grabar la clase.

## Índice

---

### Introducción a Context API

- Creación de Context API para el manejo de estado global.
- Uso de useContext para compartir datos entre componentes.
- Implementación del estado global para el carrito de compras.

### Autenticación de usuarios

- **Introducción a la Autenticación de Usuarios**
- **Implementación de formulario de login.**
- **Manejo de autenticación con tokens (simulada).**
- **Protección de rutas usando Context API para la autenticación.**

### Creación de Productos - Formulario y Validación

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

# Objetivos de la Clase

## **1 Context API**

Aprender a usar Context API para manejar el estado de autenticación.

## **2 Formulario de Login**

Implementar un formulario básico de login para autenticar usuarios.

## **3 Tokens Simulados**

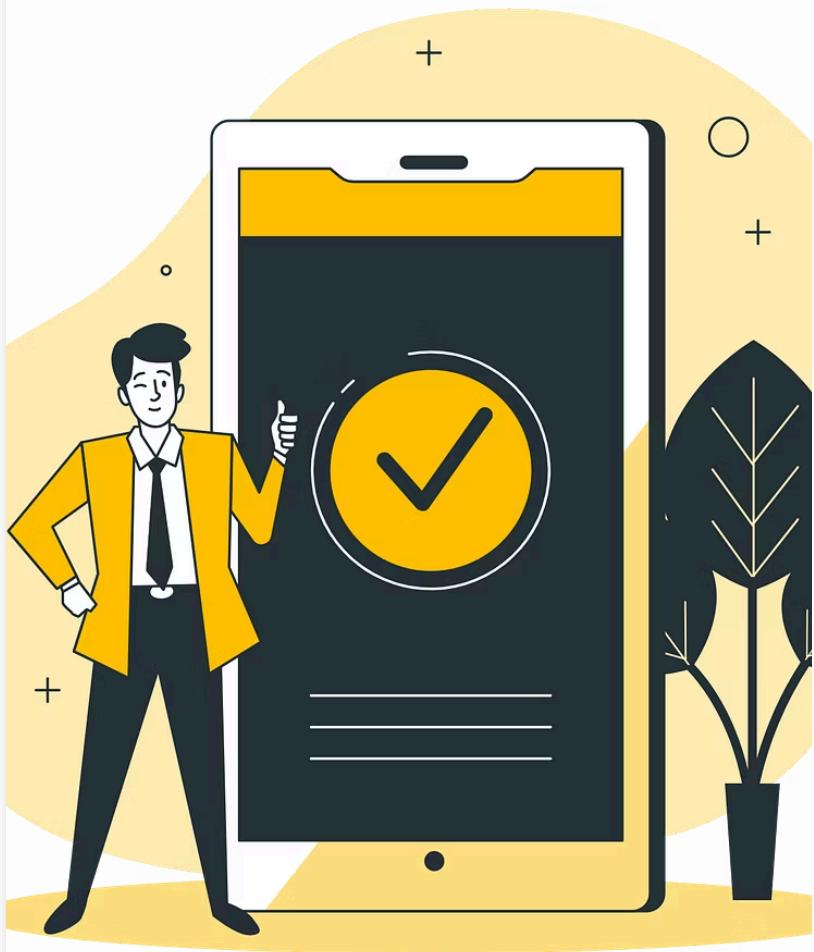
Manejar la autenticación mediante tokens simulados.

## **4 Protección de Rutas**

Proteger rutas usando Context API para la autenticación.

# Autenticación de Usuarios

---



# ¿Qué es la Autenticación?

---

La autenticación es el proceso de verificar la identidad de un usuario en una aplicación.

## Proceso Típico

Generalmente se realiza mediante un formulario de login donde el usuario ingresa sus credenciales.

# Importancia de Proteger Rutas

---

## 1 Seguridad

Evita accesos no autorizados a secciones privadas de la aplicación.

## 2 Personalización

Permite mostrar contenido específico para usuarios autenticados.

## 3 Experiencia de Usuario

Mejora la experiencia al dirigir a los usuarios a las áreas correctas.



# **Implementación de Formulario de Login**

---

# Formulario de Login

## Crear Componente

Desarrollar un componente React para el formulario de login.

## Manejo de Envío

Implementar función handleSubmit para procesar el envío del formulario.

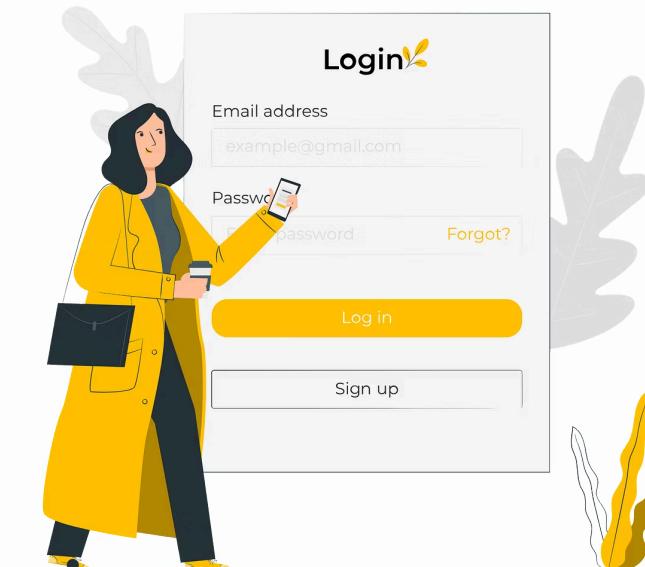
1

2

3

## Estado Local

Usar useState para manejar los valores de usuario y contraseña.



# Estructura del Componente Login

```
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;
```

# **Manejo de Autenticación con Tokens (Simulada)**

---

# Simulación de Autenticación



# Manejo de Tokens Simulados

---

## 1 Creación de Token

Generar un token simulado al autenticar al usuario.

## 2 Almacenamiento

Guardar el token en localStorage para persistencia.



# Context API para Autenticación

## 1 Crear Contexto

Definir AuthContext para manejar el estado global de autenticación.

## 2 Proveedor de Contexto

ImplementarAuthProvider para envolver la aplicación.

## 3 Hook Personalizado

Crear useAuthContext para acceder fácilmente al estado de autenticación.

# Estructura del Componente AuthContext

```
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>
  );
}
```

```
</AuthContext.Provider> );  
}  
export const useAuthContext = () => useContext(AuthContext);
```

## Protección de Rutas usando Context API

---



# Protección de Rutas

---

1

## Componente ProtectedRoute

Crear un componente para envolver rutas protegidas.

---

2

## Verificación de Autenticación

Comprobar si el usuario está autenticado.

---

3

## Redirección

Redirigir a login si no está autenticado.

# Implementación de ProtectedRoute

---

```
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;
```

# Integración en la Aplicación

## 1 Envolver con AuthProvider

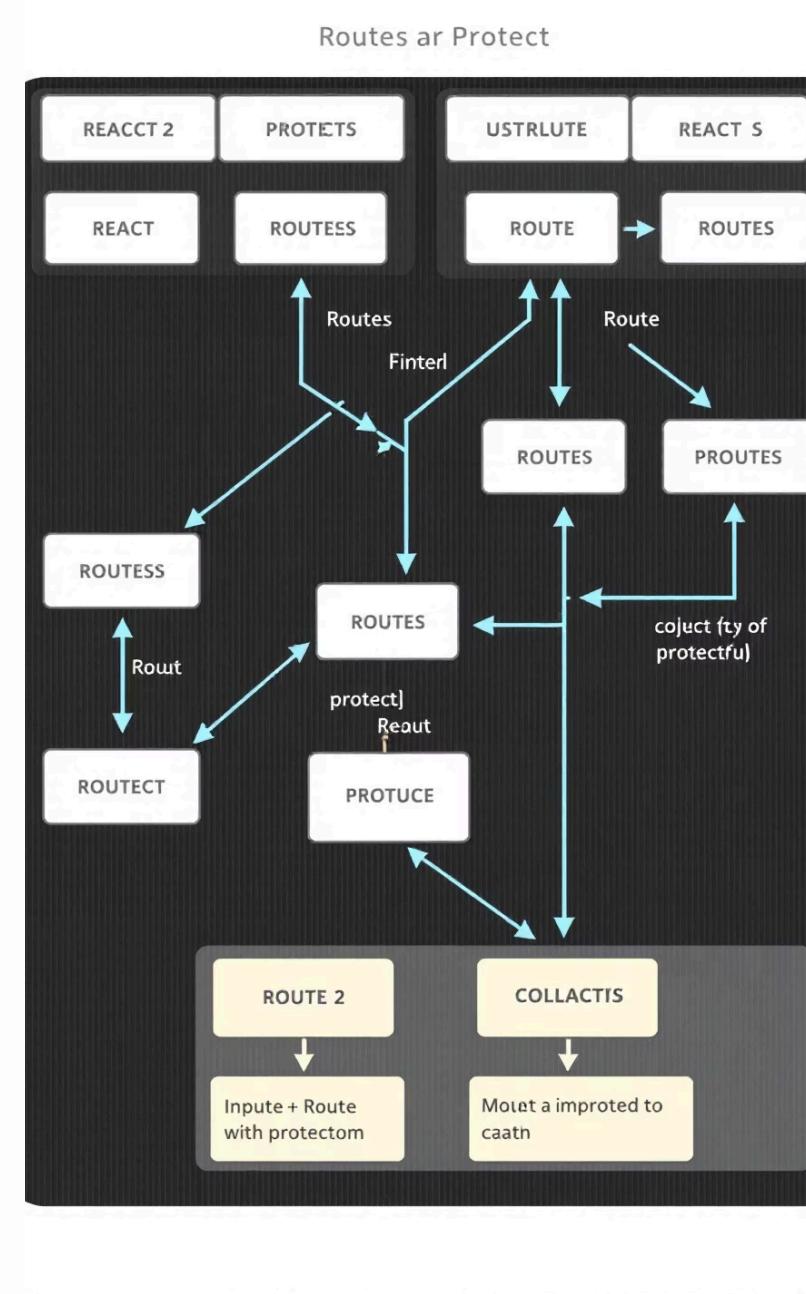
Aplicar AuthProvider al componente principal.

## 2 Definir Rutas

Configurar rutas en App.js usando React Router.

## 3 Proteger Rutas

Usar ProtectedRoute para rutas que requieren autenticación.



# Estructura de App.js

```
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;
```

```
      </Routes>
    </Router>
  </AuthProvider>
);
}

export default App;
```

# 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.



# Preguntas para Reflexionar

---



## 1 Seguridad de Tokens

¿Por qué es más seguro manejar la autenticación con tokens?

## 2 Expiración de Tokens

¿Cómo manejarías la expiración de tokens y la reautenticación?

## 3 Consideraciones de Seguridad

¿Qué aspectos de seguridad son cruciales al usar tokens en producción?

# Próximos Pasos

1

## Formulario de Productos

Crear formulario para agregar nuevos productos.

2

## Validación de Datos

Implementar validación en formularios de la aplicación.

3

## Agregar Productos

Desarrollar funcionalidad para añadir productos al catálogo.

# Recursos Adicionales

---



## Documentación de ContextAPI

[Documentación oficial de Context API](#)



## Documentación de React Router

[Documentación oficial de React Router](#)



# Ejercicios Prácticos

---



## TalentoLab – Proyecto Final

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.



# Ejercicio Práctico

Obligatorio



**Silvia**

Product Owner

**Descripción de tu tarea:**

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.



# Ejercicio Práctico

Obligatorio

## 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.



# Ejercicio Práctico

Obligatorio

## 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.