

# Entrega 2 - grupo 4

septiembre 2024

Geronimo Correa  
Tomás Peña  
Felipe Rivero

## Listado

**1. Crear un nuevo proyecto usando npx e instalar las dependencias.**

(1 Punto)

**2. Utilizar React Router para la navegación entre rutas.**

(3 Puntos)

**3. Reutilizar componentes haciendo uso del paso de props y renderización dinámica.**

(1 Punto)

**4. Hacer uso de los hooks para un desarrollo ágil y reactivo.**

(2 Puntos)

**5. Utilizar Context para el manejo del estado global.**

(3 Puntos)

# 1) Crear un nuevo proyecto usando npx e instalar las dependencias.

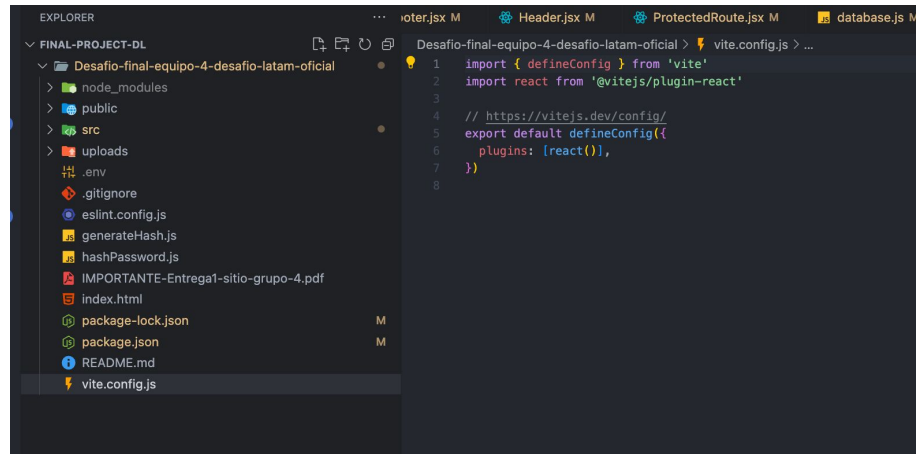
## Descripción:

Iniciamos el proyecto utilizando Vite, una herramienta de construcción moderna que permite un arranque más rápido y una experiencia de desarrollo optimizada en comparación con Create React App.

Para crear el proyecto, ejecutamos:

`npm create vite@latest`

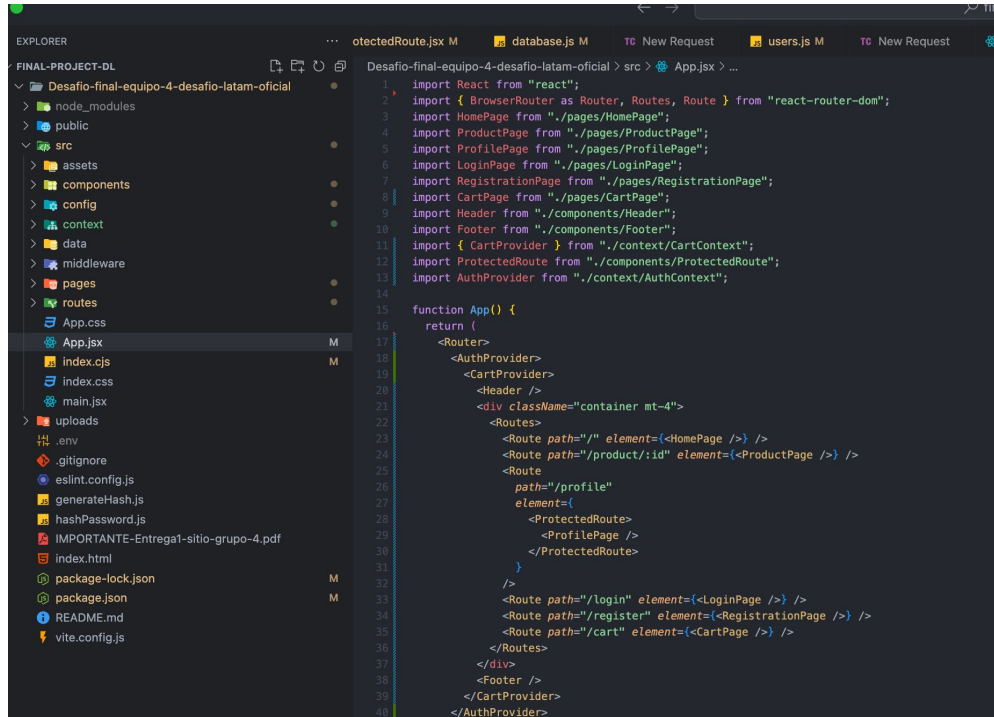
Seleccionamos React como el marco de trabajo y posteriormente instalamos todas las dependencias necesarias como React Router, Axios, y Context API para la gestión del estado global.



## 2) Utilizar React Router para la navegación entre rutas.

Descripción:

La prueba es nuestro componente App.jsx



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the project structure, including a 'src' directory with files like 'App.jsx', 'index.cjs', 'index.css', 'main.jsx', and 'uploads'. The code editor displays the content of 'App.jsx', which imports various components and providers from 'react' and 'react-router-dom', and defines a function 'App()' that returns a JSX structure for routing.

```
1 import React from "react";
2 import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
3 import HomePage from "../pages/HomePage";
4 import ProductPage from "../pages/ProductPage";
5 import ProfilePage from "../pages/ProfilePage";
6 import LoginPage from "../pages/LoginPage";
7 import RegistrationPage from "../pages/RegistrationPage";
8 import CartPage from "../pages/CartPage";
9 import Header from "../components/Header";
10 import Footer from "../components/Footer";
11 import { CartProvider } from "../context/CartContext";
12 import ProtectedRoute from "../components/ProtectedRoute";
13 import AuthProvider from "../context/AuthContext";
14
15 function App() {
16   return (
17     <Router>
18       <AuthProvider>
19         <CartProvider>
20           <Header />
21           <div className="container mt-4">
22             <Routes>
23               <Route path="/" element={<HomePage />} />
24               <Route path="/product/:id" element={<ProductPage />} />
25               <Route
26                 path="/profile"
27                 element={
28                   <ProtectedRoute>
29                     <ProfilePage />
30                   </ProtectedRoute>
31                 }
32               />
33               <Route path="/login" element={<LoginPage />} />
34               <Route path="/register" element={<RegistrationPage />} />
35               <Route path="/cart" element={<CartPage />} />
36             </Routes>
37           </div>
38           <Footer />
39         </CartProvider>
40       </AuthProvider>
41     </Router>
42   );
43 }
```

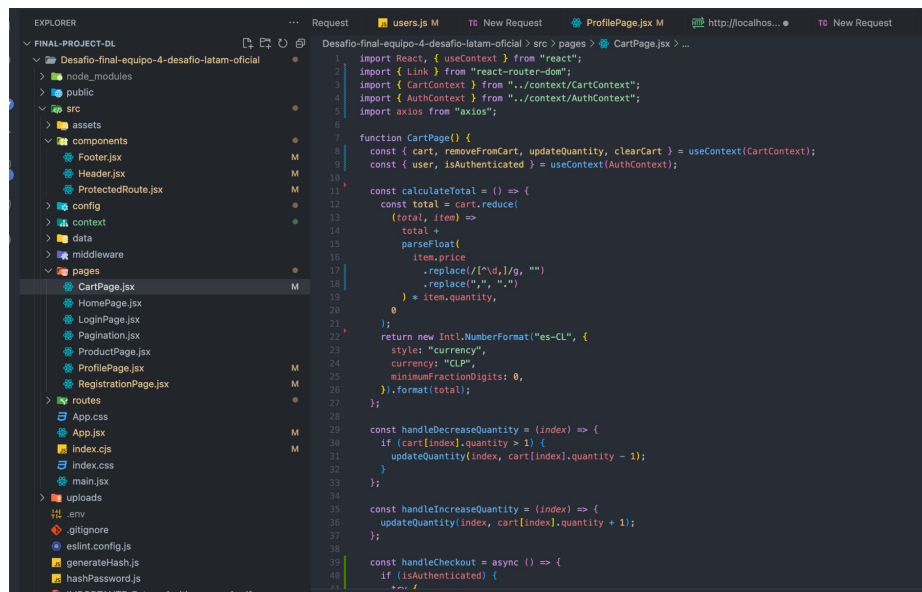


## 4) Hacer uso de los hooks para un desarrollo ágil y reactivo

### Descripción:

Utilizamos hooks como `useState`, `useEffect`, y `useContext` para manejar los cambios de estado y consumir los datos de manera reactiva.

Estos hooks nos permitieron crear componentes funcionales que se actualizan de forma eficiente sin recargar la página.



```
1 import React, { useContext } from "react";
2 import { Link } from "react-router-dom";
3 import { CartContext } from "../context/CartContext";
4 import { AuthContext } from "../context/AuthContext";
5 import axios from "axios";
6
7 function CartPage() {
8   const { cart, removeFromCart, updateQuantity, clearCart } = useContext(CartContext);
9   const { user, isAuthenticated } = useContext(AuthContext);
10
11   const calculateTotal = () => {
12     const total = cart.reduce(
13       (total, item) =>
14         total +
15         parseFloat(
16           item.price
17             .replace(/[^\d,]/g, "")
18             .replace(",", ".")
19         ) * item.quantity,
20       0
21     );
22     return new Intl.NumberFormat("es-CL", {
23       style: "currency",
24       currency: "CLP",
25       minimumFractionDigits: 0,
26     }).format(total);
27   };
28
29   const handleDecreaseQuantity = (index) => {
30     if (cart[index].quantity > 1) {
31       updateQuantity(index, cart[index].quantity - 1);
32     }
33   };
34
35   const handleIncreaseQuantity = (index) => {
36     updateQuantity(index, cart[index].quantity + 1);
37   };
38
39   const handleCheckout = async () => {
40     if (isAuthenticated) {
41       // ...
42     }
43   };
44 }
```

## 5) Utilizar Context para el manejo del estado global.

### **Descripción:**

**Implementamos Context API para manejar el estado global del usuario autenticado y el carrito de compras.**

**El contexto simplifica el paso de información entre componentes sin necesidad de prop drilling.**

**Ejemplo: El usuario autenticado es accesible desde cualquier parte de la aplicación a través de AuthContext.**

**Captura: Código del AuthContext y cómo se consume en diferentes componentes.**