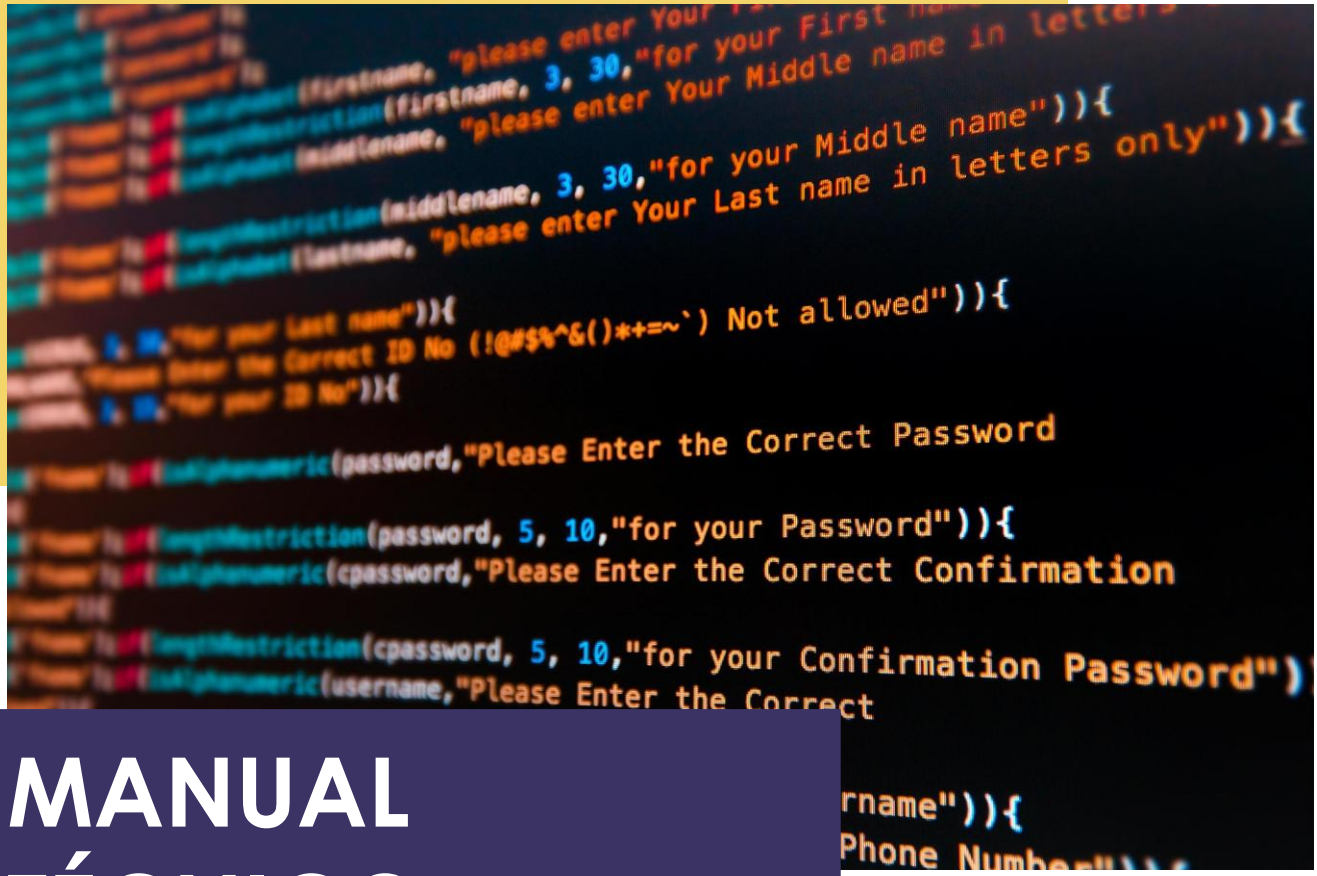


MANUAL TÉCNICO

dojo de karate "Cobra"

Curso	Carné	Nombre
IPC1	201907179	Wilson Manuel Santos Ajcot

TABLA DE CONTENIDO

1. INTRODUCCIÓN

El sistema de Gestión de Productos y Clientes es una solución integral diseñada para optimizar la administración de inventarios y bases de datos de clientes. Este sistema ofrece una interfaz web intuitiva para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) y proporciona gráficas interactivas para el análisis de datos.

El desarrollo del sistema combina tecnologías modernas como React en el frontend y Node.js en el backend, garantizando escalabilidad y facilidad de mantenimiento

2. OBJETIVOS

Objetivo General:

Facilitar la gestión de productos y clientes mediante una herramienta web eficiente, segura e intuitiva.

Objetivos Específicos:

- Proporcionar una documentación técnica detallada para desarrolladores y administradores.
- Garantizar la integración segura entre el frontend y el backend.
- Facilitar el análisis de datos mediante gráficas dinámicas.
- Ofrecer un diseño modular que permita futuras expansiones y mejoras.

3. DIRIGIDO

Este manual está orientado a:

- **Desarrolladores:** Que deseen mantener o extender la funcionalidad del sistema.
- **Administradores del Sistema:** Responsables de supervisar su correcto funcionamiento.
- **Usuarios Técnicos:** Interesados en implementar soluciones similares.

4. ESPECIFICACIÓN TÉCNICA

4.1 REQUISITOS DEL HARDWARE

- Computadora de escritorio o portátil.
- Memoria RAM: Mínimo 4 GB.
- Almacenamiento: Disco duro de 250 GB o superior.
- Procesador Intel Core i3 o superior
- Resolución Gráfica: Resolución mínima de 1024 x 768 píxeles

4.2 REQUISITOS DEL SOFTWARE

- Sistema Operativo: Windows 10 o superior / Ubuntu 20.04 o superior.
- Node.js: Versión 16 o superior.
- React: Versión 18 o superior.
- Navegador web: Google Chrome, Firefox o Edge.

5. ESTRUCTURA DEL PROYECTO

El proyecto está organizado en los siguientes módulos principales:

5.1 Frontend (React):

Componentes para la interfaz de usuario, como el login, el dashboard y las gráficas.
Conexión con el backend mediante Axios para realizar peticiones HTTP.

5.1.1. App.jsx

Este archivo es el punto de entrada principal de las rutas de la aplicación. Utiliza react-router-dom para definir rutas a las páginas principales, como el Login y el Dashboard. Implementa protección mediante el componente ProtectedRoute para restringir el acceso al Dashboard solo a usuarios autenticados.

```jsx
import { Routes, Route } from "react-router-dom";
import Login from "./pages/Login";
import Dashboard from "./pages/Dashboard";

import ProtectedRoute from "./components/ProtectedRoute";

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

export default App;
```

5.1.2. Dashboard.jsx

Maneja la interfaz y lógica principal del panel de control. Permite a los usuarios visualizar y gestionar productos y clientes a través de tablas, realizar operaciones CRUD, y analizar datos mediante gráficas interactivas de barras y circulares. Se conecta al backend para actualizar y obtener datos en tiempo real.

5.1.3. Login.jsx

Proporciona la interfaz de autenticación de usuarios. Incluye un formulario que envía credenciales al backend para validación y redirige al Dashboard en caso de éxito. Muestra mensajes de error si las credenciales no son válidas o hay problemas de conexión.

```jsx
import { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import { useAuth } from "../context/AuthContext";
import axios from "axios";
import "./Login.css";
import Swal from "sweetalert2";
const Login = () => {
  const [credentials, setCredentials] = useState({ username: "",
password: "" });
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);
  const { isAuthenticated, login } = useAuth();
  const navigate = useNavigate();

  // Redirige al Dashboard si ya está autenticado
  useEffect(() => {
    if (isAuthenticated) {
      navigate("/dashboard");
    }
  }, [isAuthenticated, navigate]);

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

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);
    setError("");

    try {
      const response = await axios.post(
        "http://localhost:3005/store/login",
        credentials
      );

      if (response.data.status === "success") {
        Swal.fire("Éxito", "Bienvenido", "success");
        login();
      } else {
        setError(response.data.message || "Error desconocido");
      }
    } catch (err) {
      if (err.response && err.response.status === 401) {
        // Error de autenticación
        //  setError(err.response.data.message || "");
        Swal.fire("Error", "Usuario o contraseña incorrectos", "error");

      } else {
        // Error de conexión o servidor
        setError("Error al conectar con el servidor");
      }
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="login-container">
      <form onSubmit={handleSubmit} className="login-form">
        <h2 className="login-title">Iniciar Sesión</h2>
        {error && <p className="login-error">{error}</p>}
        <div className="login-field">
          <label>Usuario:</label>
          <input
            type="text"
            name="username"
            value={credentials.username}
            onChange={handleInputChange}
            placeholder="Ingresa tu usuario"
            required
          />
        </div>
        <div className="login-field">
          <label>Contraseña:</label>
          <input
            type="password"
            name="password"
            value={credentials.password}
            onChange={handleInputChange}
            placeholder="Ingresa tu contraseña"
            required
          />
        </div>
        <button type="submit" className="login-button" disabled=
{loading}>
          {loading ? "Cargando..." : "Iniciar Sesión"}
        </button>
      </form>
    </div>
  );
};
export default Login;
```

5.1.4. ProtectedRoute.jsx

Garantiza que solo usuarios autenticados puedan acceder a ciertas rutas (como el Dashboard). Verifica el estado de autenticación desde el contexto global (AuthContext) antes de permitir el acceso.

```jsx
import { Navigate } from "react-router-dom";
import { useAuth } from "../context/AuthContext";
import PropTypes from "prop-types";

const ProtectedRoute = ({ children }) => {
  const { isAuthenticated } = useAuth();

  if (!isAuthenticated) {
    return <Navigate to="/" replace />;
  }

  return children;
};

ProtectedRoute.propTypes = {
  children: PropTypes.node.isRequired,
};

export default ProtectedRoute;
```

5.1.5. AuthContext.jsx

Gestiona el estado global de autenticación utilizando el Context API de React. Proporciona funciones para iniciar sesión (login) y cerrar sesión (logout), y un estado booleano (isAuthenticated) para verificar si el usuario está autenticado.

```jsx
import { createContext, useContext, useState } from "react";
import PropTypes from "prop-types";
import Swal from "sweetalert2";
const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(() =>
    localStorage.getItem("isAuthenticated") === "true"
  );

  const login = () => {
    setIsAuthenticated(true);
    localStorage.setItem("isAuthenticated", "true");
  };

  const logout = () => {
    setIsAuthenticated(false);
    localStorage.removeItem("isAuthenticated");
    Swal.fire("Éxito", "Vuelva Pronto", "success")
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};

AuthProvider.propTypes = {
  children: PropTypes.node.isRequired,
};

export const useAuth = () => useContext(AuthContext);
```

5.1.6. Dashboard.css

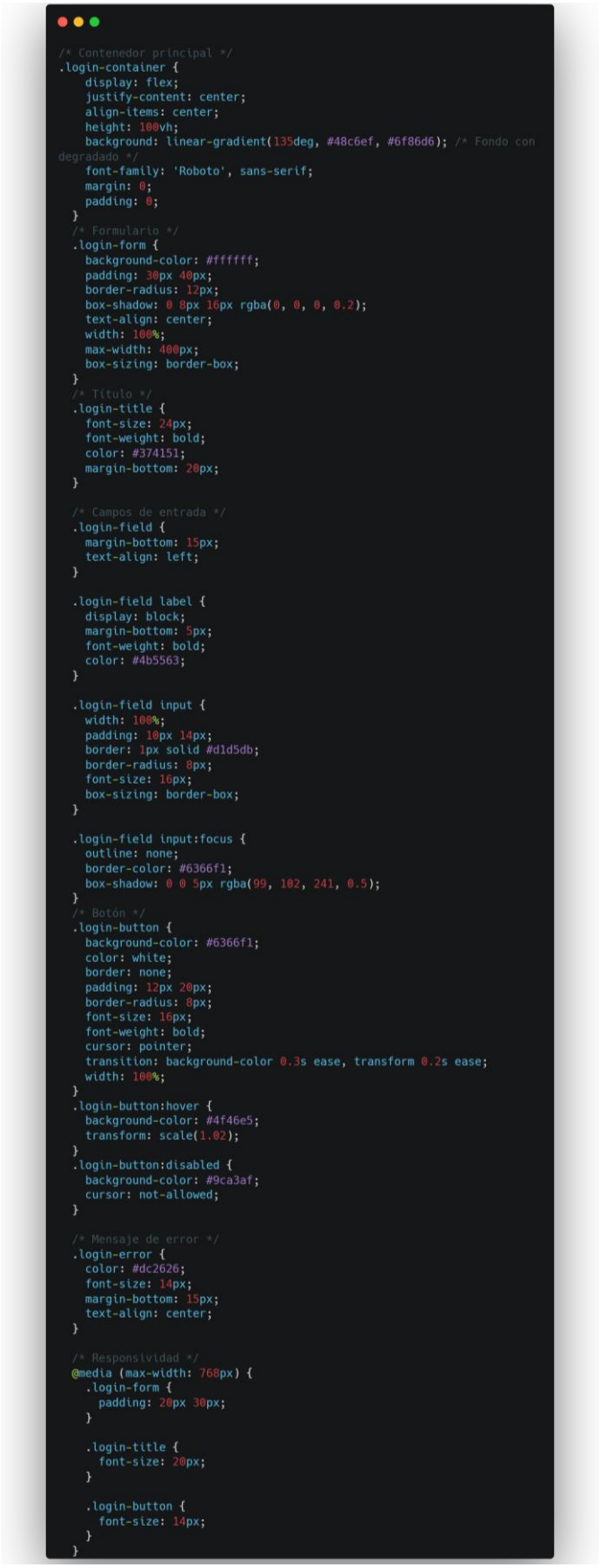
Define los estilos personalizados para el Dashboard, incluyendo el diseño de tablas, formularios, botones, y gráficas. Utiliza colores y efectos visuales para mejorar la experiencia de usuario y mantener una estética profesional.

```css
body {
    font-family: 'Roboto', sans-serif;
    background-color: #f4f7fc;
    margin: 0;
    padding: 0;
    overflow-x: hidden; /* Evita scroll horizontal */
}

.dashboard-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    padding: 20px;
    gap: 30px;
    max-width: 1200px; /* Centra el contenido */
    margin: auto; /* Agrega margen automático */
    box-sizing: border-box;
}

.dashboard-header {
    width: 100%;
    background-color: #3f51b5;
    color: white;
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 15px 20px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    border-radius: 8px;
}

.dashboard-header h1 {
    margin: 0;
    font-size: 24px;
}

.logout-button {
    background-color: #e53935;
    color: white;
    border: none;
    padding: 10px 15px;
    border-radius: 5px;
    font-size: 14px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

.logout-button:hover {
    background-color: #c62828;
}

.dashboard-content {
    display: flex;
    flex-direction: column;
    gap: 30px;
    width: 100%;
    box-sizing: border-box;
}

.form-container {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    gap: 20px;
    width: 100%;
}

.form-section {
    background-color: #ffffff;
    border-radius: 12px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
    padding: 20px;
    flex: 1;
    min-width: 280px;
    max-width: 400px;
}

.form-section h2 {
    font-size: 20px;
    color: #333;
    margin-bottom: 15px;
    text-align: center;
}

.form-section input {
    width: 100%;
    margin-bottom: 15px;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 8px;
    font-size: 16px;
}

.form-section input:focus {
    border-color: #3f51b5;
    outline: none;
    box-shadow: 0 0 5px rgba(63, 81, 181, 0.3);
}
```

```css
.btn-submit {
    width: 100%;
    background-color: #4caf50;
    color: white;
    padding: 10px 15px;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-size: 16px;
    font-weight: bold;
    transition: background-color 0.3s ease;
}

.btn-submit:hover {
    background-color: #388e3c;
}

.charts-section,
.tables-section {
    width: 100%;
    background-color: #ffffff;
    border-radius: 12px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
    padding: 20px;
}

.chart {
    width: 100%;
    height: 400px; /* Fija la altura para evitar desbordes */
    margin-bottom: 30px;
}

.tables-section table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}

.tables-section th {
    background-color: #3f51b5;
    color: white;
    padding: 12px;
    text-align: left;
    font-size: 16px;
}

.tables-section td {
    padding: 12px;
    border: 1px solid #ddd;
    font-size: 14px;
}

.tables-section tr:nth-child(even) {
    background-color: #f4f7fc;
}

.tables-section tr:hover {
    background-color: #e8f0fe;
}

.btn-delete {
    background-color: #e53935;
    color: white;
    border: none;
    padding: 5px 10px;
    border-radius: 5px;
    cursor: pointer;
    font-size: 14px;
}

.btn-delete:hover {
    background-color: #c62828;
}

/* Ajuste responsivo */
@media (max-width: 768px) {
    .form-container {
        flex-direction: column;
        align-items: center;
    }

    .charts-section .chart {
        height: 300px; /* Ajuste de altura en dispositivos pequeños */
    }
}
.chart {
    width: 100%;
    max-width: 800px; /* Ajusta según tu diseño */
    height: 400px; /* Fija la altura para gráficos claros */
    margin: auto; /* Centrado */
}
```

5.1.7. Login.css

Proporciona estilos específicos para la página de Login, como diseño del formulario, estilos para mensajes de error y botones. Mantiene la coherencia visual con el resto de la aplicación.



```css
/* Contenedor principal */
.login-container {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background: linear-gradient(135deg, #48c6ef, #6f86d6); /* Fondo con
degradado */
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
  }
  /* Formulario */
  .login-form {
    background-color: #ffffff;
    padding: 30px 40px;
    border-radius: 12px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
    text-align: center;
    width: 100%;
    max-width: 400px;
    box-sizing: border-box;
  }
  /* Título */
  .login-title {
    font-size: 24px;
    font-weight: bold;
    color: #374151;
    margin-bottom: 20px;
  }

  /* Campos de entrada */
  .login-field {
    margin-bottom: 15px;
    text-align: left;
  }

  .login-field label {
    display: block;
    margin-bottom: 5px;
    font-weight: bold;
    color: #4b5563;
  }

  .login-field input {
    width: 100%;
    padding: 10px 14px;
    border: 1px solid #d1d5db;
    border-radius: 8px;
    font-size: 16px;
    box-sizing: border-box;
  }

  .login-field input:focus {
    outline: none;
    border-color: #6366f1;
    box-shadow: 0 0 5px rgba(99, 102, 241, 0.5);
  }
  /* Botón */
  .login-button {
    background-color: #6366f1;
    color: white;
    border: none;
    padding: 12px 20px;
    border-radius: 8px;
    font-size: 16px;
    font-weight: bold;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease;
    width: 100%;
  }
  .login-button:hover {
    background-color: #4f46e5;
    transform: scale(1.02);
  }
  .login-button:disabled {
    background-color: #9ca3af;
    cursor: not-allowed;
  }

  /* Mensaje de error */
  .login-error {
    color: #dc2626;
    font-size: 14px;
    margin-bottom: 15px;
    text-align: center;
  }

  /* Responsividad */
  @media (max-width: 768px) {
    .login-form {
      padding: 20px 30px;
    }

    .login-title {
      font-size: 20px;
    }

    .login-button {
      font-size: 14px;
    }
  }
```

5.1.8. App.css

Contiene estilos generales compartidos en toda la aplicación, como el diseño de la página principal, fuentes, colores base, y espaciado global.

```css
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}

@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}

.card {
  padding: 2em;
}

.read-the-docs {
  color: #888;
}
```

5.2 Backend (Node.js):

Endpoints para gestionar productos, clientes y autenticación.

Middleware para validar solicitudes y gestionar errores.

5.2.1. config.js:

Contiene la configuración global del sistema, como las credenciales predeterminadas del administrador (adminData) y la configuración del servidor (server). Centraliza parámetros que pueden ser modificados según las necesidades del entorno de desarrollo o producción.

```
const dotenv =require('dotenv');
dotenv.config();


module.exports={
    server:{
        port:process.env.PORT
    },
    adminData:{
        password:process.env.PASSWORD,
        username:process.env.USERSTORE
    }
}
n go(f, seed, [])
}
```

5.2.2. loginController.js:

Implementa la lógica del login. Valida las credenciales ingresadas por el usuario comparándolas con las almacenadas en la configuración. Si las credenciales son correctas, responde con un estado exitoso; de lo contrario, devuelve un mensaje de error con un estado HTTP adecuado.

```
const { adminData } = require("../config/config");

// Controlador para manejar la autenticación
const login = async (req, res) => {
    try {
        const { username, password } = req.body;

        if (username === adminData.username && password === adminData.password) {
            return res.status(200).json({
                message: "Inicio de sesión exitoso",
                status: "success",
            });
        }

        return res.status(401).json({
            message: "Usuario o contraseña incorrectos, intente de nuevo",
            status: "error",
        });
    } catch (error) {
        return res.status(500).json({
            message: "Error interno del servidor",
            error: error.message,
        });
    }
};

module.exports = { login };
```

5.2.3. productController.js:

Gestiona las operaciones CRUD relacionadas con los productos. Permite crear, leer, actualizar y eliminar productos en la base de datos. Sirve como la capa lógica que interactúa con los modelos de datos y devuelve las respuestas necesarias al frontend.

```
let nextProductId = 1;
const products = []; // Base de datos simulada para productos

const addProduct = (req, res) => {
  const { name, price, stock } = req.body;

  // Validaciones
  if (!name || typeof name !== "string" || name.trim() === "") {
    return res.status(400).json({
      message: "Error en los datos del producto: el nombre es requerido",
      status: "error",
    });
  }

  if (!price || typeof price !== "number" || price < 0) {
    return res.status(400).json({
      message: "Error en los datos del producto: el precio debe ser mayor que 0 y numérico",
      status: "error",
    });
  }

  if (stock === undefined || typeof stock !== "number" || stock <=0) {
    return res.status(400).json({
      message: "Error en los datos del producto: el stock debe ser mayor  a 0 y numérico",
      status: "error",
    });
  }

  // Validar duplicados por nombre
  if (products.some((p) => p.name === name)) {
    return res.status(400).json({
      message: "Error en los datos del producto: el nombre del producto ya existe",
      status: "error",
    });
  }

  // Generar ID autoincremental
  const id = nextProductId++;

  products.push({ id, name, price, stock });
  return res.status(200).json({
    message: "Producto agregado exitosamente",
    status: "success",
    product: { id, name, price, stock },
  });
};

const getProducts = (req, res) => {
  return res.status(200).json(products);
};

const deleteProduct = (req, res) => {
  const { id } = req.params;

  const index = products.findIndex((product) => product.id === parseInt(id, 10));

  if (index === -1) {
    return res.status(404).json({
      message: "Producto no encontrado",
      status: "error",
    });
  }

  products.splice(index, 1);

  // Reiniciar ID si todos los productos son eliminados (opcional)
  if (products.length === 0) {
    nextProductId = 1;
  }

  return res.status(200).json({
    message: "Producto eliminado exitosamente",
    status: "success",
  });
};
module.exports = { addProduct, getProducts, deleteProduct };
```

5.2.4. clientController.js

Similar al productController.js, este archivo maneja las operaciones CRUD, pero enfocadas en la gestión de clientes. Valida las entradas y asegura que las operaciones sean procesadas correctamente, devolviendo resultados apropiados.

```
let nextId = 1;
const clients = []; // Base de datos simulada

const addClient = (req, res) => {
  const { name, age, nit } = req.body;

  if (!name || typeof name !== "string" || name.trim() === "") {
    return res.status(400).json({
      message: "Error en los datos del cliente: el nombre es requerido",
      status: "error",
    });
  }

  if (!age || typeof age !== "number" || age <= 0) {
    return res.status(400).json({
      message:
        "Error en los datos del cliente: la edad debe ser mayor que 0 y numérica",
      status: "error",
    });
  }

  const id = nextId++; // Generar ID autoincremental

  clients.push({ id, name, age, nit: nit || "C/F" });
  return res.status(200).json({
    message: "Cliente agregado exitosamente",
    status: "success",
    client: { id, name, age, nit: nit || "C/F" },
  });
};

const getClients = (req, res) => {
  return res.status(200).json(clients);
};

const deleteClient = (req, res) => {
  const { id } = req.params;

  const index = clients.findIndex((client) => client.id === parseInt(id, 10));

  if (index === -1) {
    return res.status(404).json({
      message: "Cliente no encontrado",
      status: "error",
    });
  }

  clients.splice(index, 1);

  // Reiniciar ID si todos los clientes son eliminados
  if (clients.length === 0) {
    nextId = 1;
  }

  return res.status(200).json({
    message: "Cliente eliminado exitosamente",
    status: "success",
  });
};

module.exports = { addClient, getClients, deleteClient };
```

5.2.5. clients.js:

Modelo o esquema para los datos de clientes. Define cómo se estructuran y almacenan los datos. Incluye campos como nombre, edad y NIT.

```
const clients = [];

module.exports = clients;
```

5.2.6. products.js:

Modelo para los productos, definiendo su estructura en la base de datos. Incluye campos como nombre, precio y stock. Es utilizado por el controlador de productos para interactuar con la base de datos.

```
const products = [];

module.exports = products;
```

5.2.7. clientRoutes.js:

Define las rutas específicas para las operaciones CRUD de clientes. Conecta las solicitudes HTTP (GET, POST, PUT, DELETE) con las funciones correspondientes en el clientController.

```
const express = require("express");
const { addClient, getClients,deleteClient  }
= require("../controllers/clientController");

const router = express.Router();

router.post("/store/clients", addClient);
router.get("/store/clients", getClients);
// Endpoint para eliminar un cliente
router.delete("/store/clients/:id",
deleteClient);
module.exports = router;
```

5.2.8. loginRoutes.js

Contiene las rutas relacionadas con la autenticación. Procesa solicitudes para la operación de login y las delega al loginController. Garantiza que los endpoints sean claros y específicos para este propósito.

```
const express = require("express");
const { login } =
require("../controllers/loginController");

const router = express.Router();

// Ruta para el login
router.post("/store/login", login);

module.exports = router;
```

5.2.9. productRoutes.js:

Similar a clientRoutes.js, este archivo gestiona las rutas HTTP que interactúan con el productController. Organiza endpoints como /products para asegurar una gestión ordenada de los productos.

```
const express = require("express");
const { addProduct, getProducts, deleteProduct }
= require("../controllers/productController");

const router = express.Router();
//endpoint para agregar productos
router.post("/store/products", addProduct);
//endpoint para obtener los productos
router.get("/store/products", getProducts);
// Endpoint para eliminar un producto
router.delete("/store/products/:id",
deleteProduct);
module.exports = router;
```

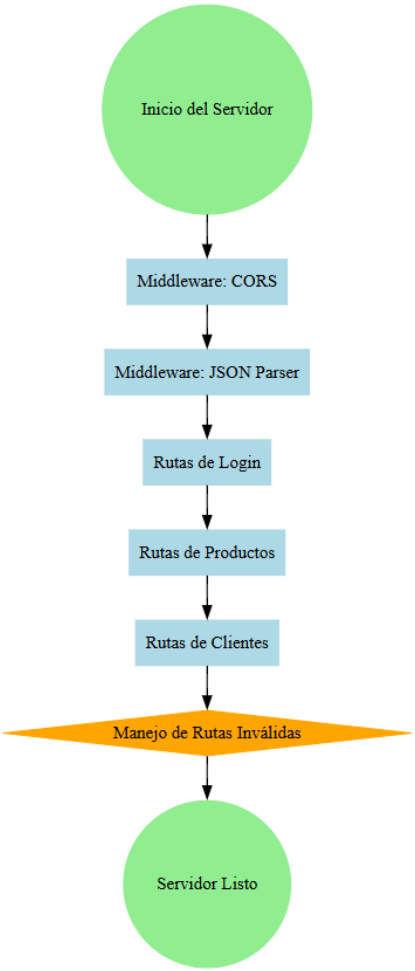
5.2.10. 404.js:

Archivo para manejar rutas no encontradas. Responde con un estado 404 y un mensaje descriptivo cuando un usuario intenta acceder a una ruta que no está definida en la aplicación.

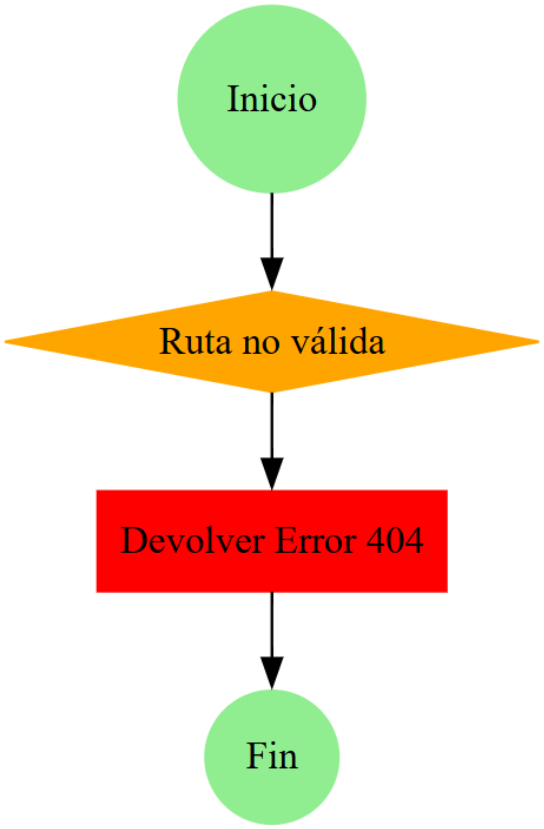
```
const express = require("express");

const router = express.Router();

router.use((req, res, next) => {
  res.status(404).json({
    message: "Invalid endpoint",
  });
});
module.exports = router;
```

6. DIAGRAMAS DE FLUJO

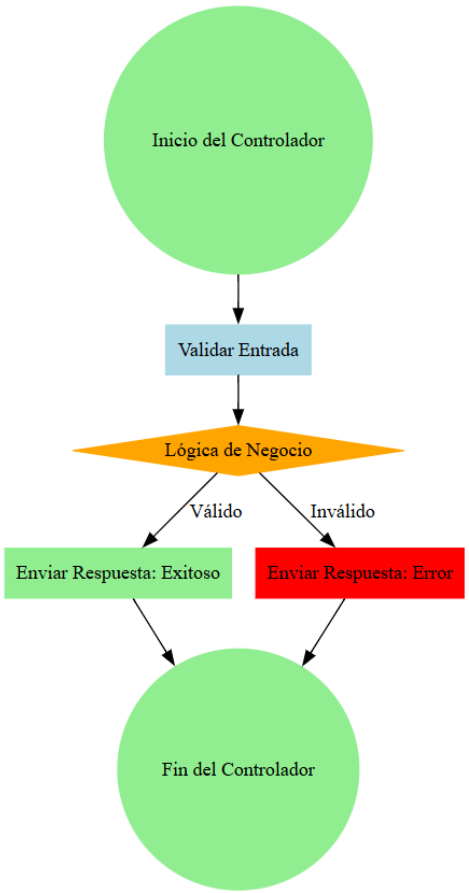
LOGICA DEL INDEX



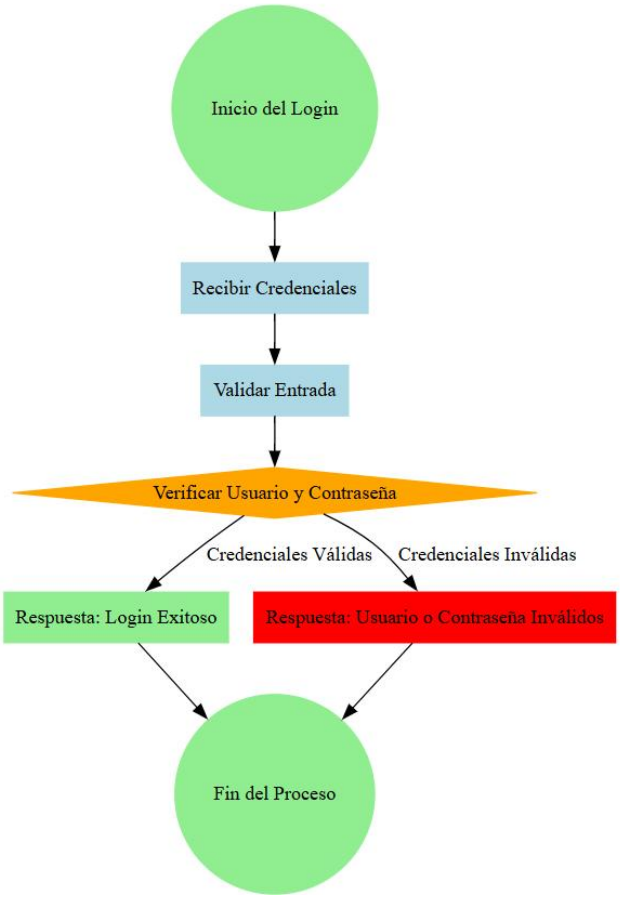
- **LÓGICA DE RUTAS**



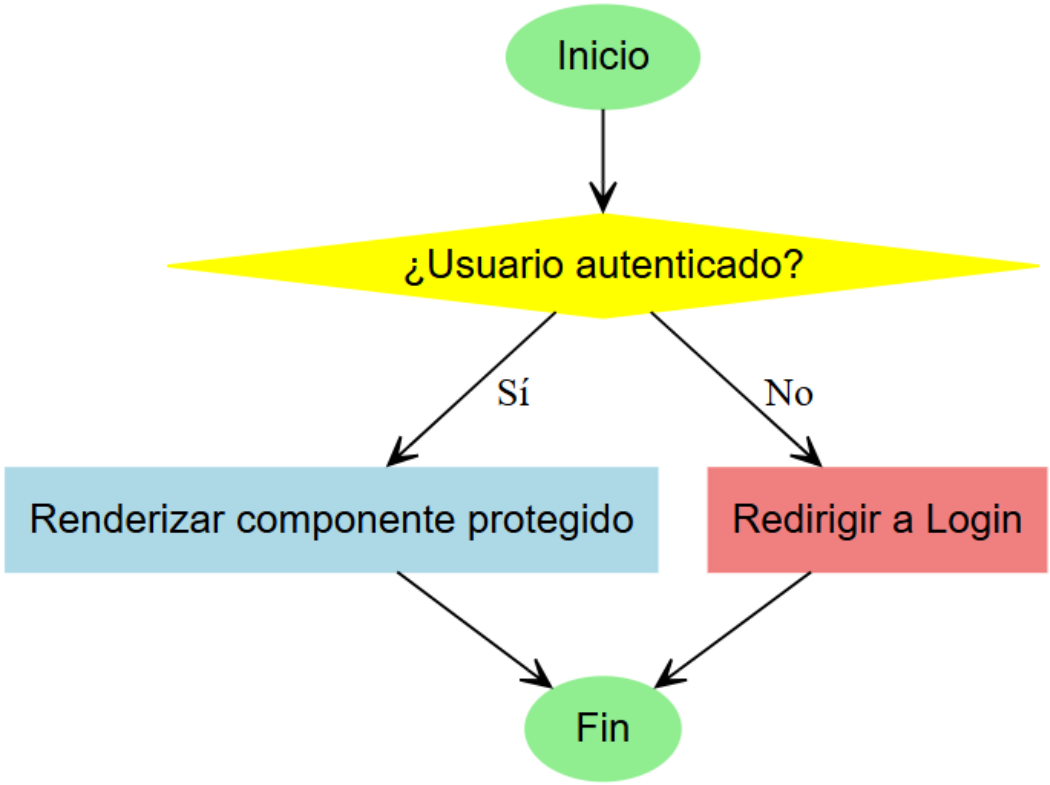
LOGICA DEL CONTROLADOR



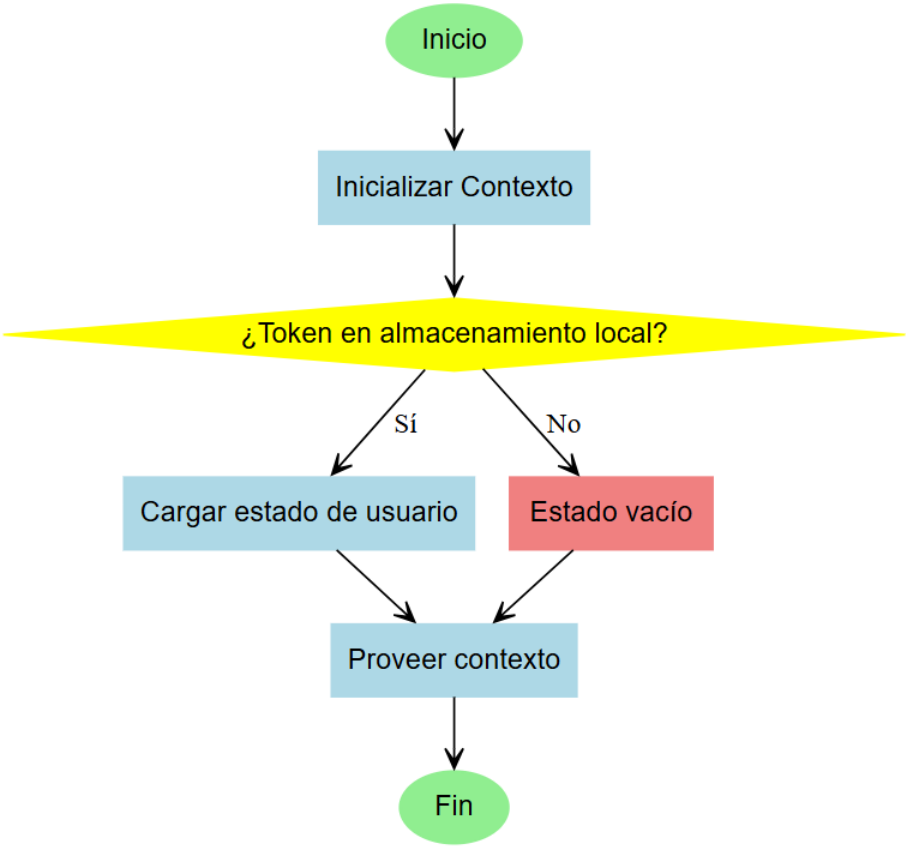
LOGICA DEL LOGIN



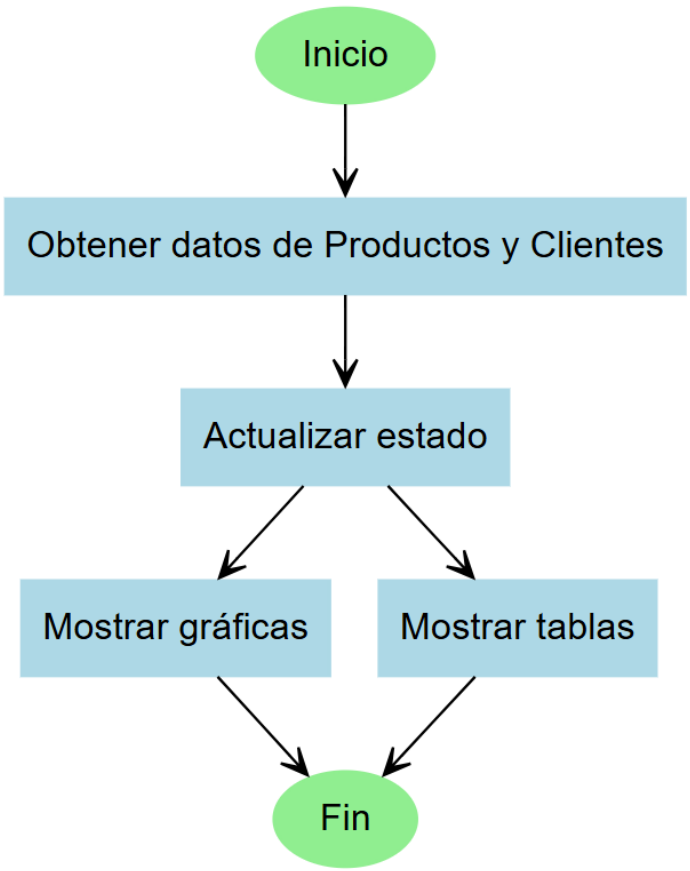
ProtectedRoute.jsx



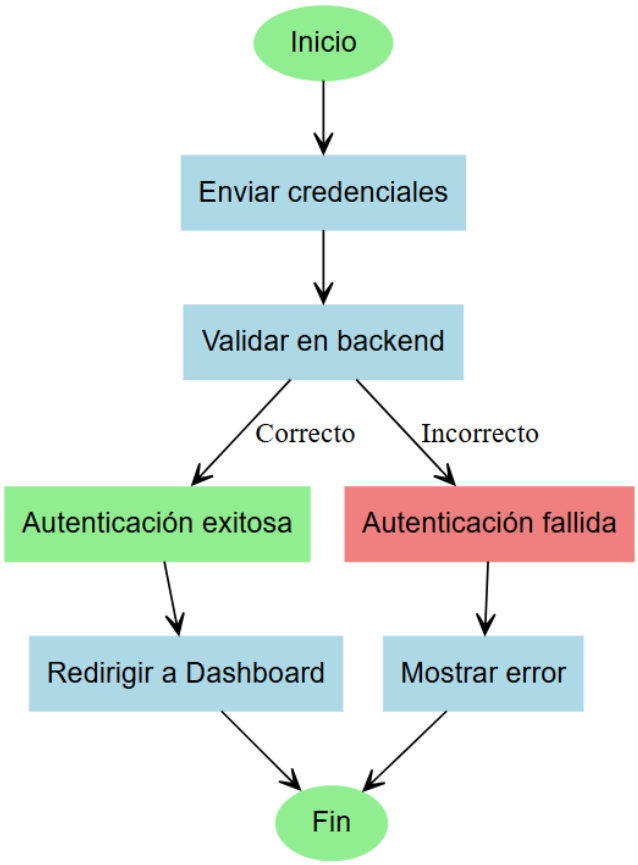
AuthContext.jsx



Dashboard.jsx



Login.jsx


Inicio
Enviar credenciales
Validar en backend
Correcto
Incorrecto
Autenticación exitosa
Autenticación fallida
Redirigir a Dashboard
Mostrar error
Fin