**PROYECTO**

(React,Mongo Atlas,

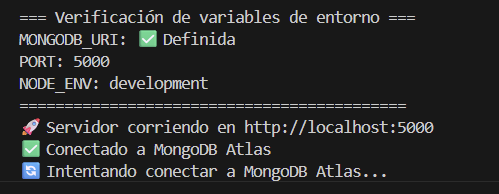
Node y Express)

**Nombre del Proyecto:** Clínica Serenity

**Descripción:**es una aplicación web de gestión integral diseñada para clínicas y consultorios médicos. El sistema facilita la administración de información crucial sobre pacientes, médicos y citas, permitiendo un control eficiente de los datos operativos diarios.

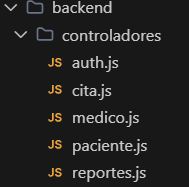
Además de las funcionalidades básicas de creación, lectura, actualización y eliminación (CRUD) de registros,

el proyecto incorpora una característica clave: la generación de reportes en formato PDF.

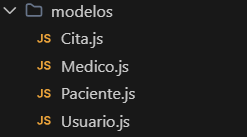
.

1. **Organización de Carpetas del Backend y Frontend**

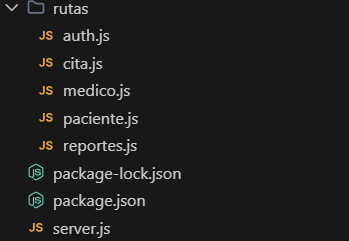
* backend/controladores/: Contiene la lógica de negocio para cada entidad. Aquí se definen las funciones que procesan las solicitudes del cliente, interactúan con la base de datos y preparan las respuestas.



* backend/modelos/: Define los esquemas y modelos de datos de MongoDB utilizando Mongoose. Cada archivo representa una colección en la base de datos.



* backend/rutas/: Define los endpoints de la API y asocia cada ruta con la función del controlador correspondiente.



* backend/middleware/: Contiene funciones intermedias que se ejecutan antes de que las solicitudes lleguen a los controladores, como la verificación de autenticación.



Auth.js

const jwt = require('jsonwebtoken');

const Usuario = require('../modelos/Usuario');

exports.verificarToken = async (req, res, next) => {

  const token = req.header('Authorization');

  if (!token) {

    return res.status(401).json({ msg: 'No token, autorización denegada' });

  }

  try {

    const tokenLimpio = token.replace('Bearer ', '');

    const decoded = jwt.verify(tokenLimpio, process.env.JWT\_SECRET);

    req.usuario = decoded.usuario;

    next();

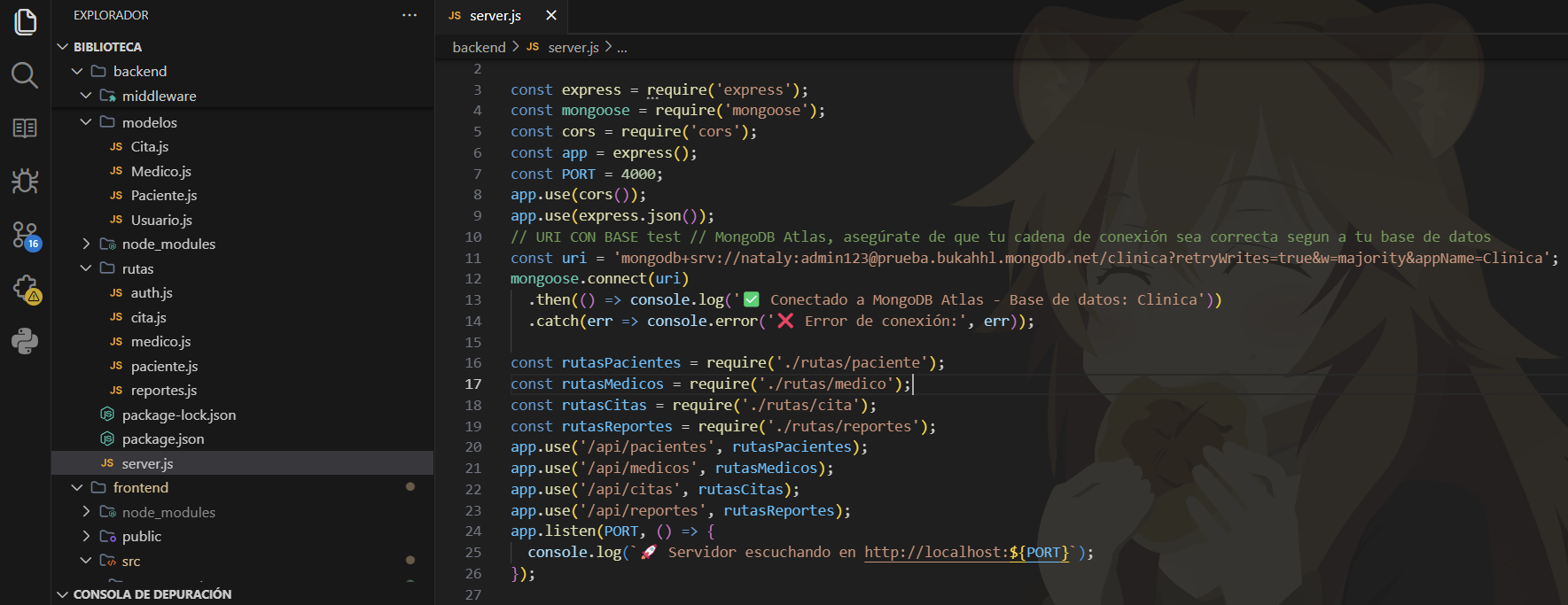
  } catch (err) {

    res.status(401).json({ msg: 'Token no válido' });

  }

};

* backend/server.js: Es el archivo principal del servidor, donde se configura Express, se establece la conexión a la base de datos y se montan todas las rutas.



Server.js

// File: backend/server.js

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const app = express();

const PORT = 4000;

app.use(cors());

app.use(express.json());

// URI CON BASE test

const uri = 'mongodb+srv://nataly:admin123@prueba.bukahhl.mongodb.net/clinica?retryWrites=true&w=majority&appName=Clinica';

mongoose.connect(uri)

  .then(() => console.log('✅ Conectado a MongoDB Atlas - Base de datos: Clinica'))

  .catch(err => console.error('❌ Error de conexión:', err));

const rutasPacientes = require('./rutas/paciente');

const rutasMedicos = require('./rutas/medico');

const rutasCitas = require('./rutas/cita');

const rutasReportes = require('./rutas/reportes');

app.use('/api/pacientes', rutasPacientes);

app.use('/api/medicos', rutasMedicos);

app.use('/api/citas', rutasCitas);

app.use('/api/reportes', rutasReportes);

app.listen(PORT, () => {

  console.log(`🚀 Servidor escuchando en http://localhost:${PORT}`);

});

**Organización del Frontend (React)**

El frontend se encuentra en la carpeta frontend/ y está estructurado para facilitar el desarrollo de componentes reutilizables.

* frontend/src/components/: Contiene los componentes de React que forman la interfaz de usuario. Cada componente es una pieza modular de la UI.



* frontend/src/: Contiene los archivos principales de la aplicación React (App.js, index.js), estilos globales y utilidades.

App.js

import { useState } from "react"

import ListaPacientes from "./components/ListaPacientes"

import ListaMedicos from "./components/ListaMedicos"

import ListaCitas from "./components/ListaCitas"

import Inicio from "./components/Inicio"

import "bootstrap/dist/css/bootstrap.min.css"

import "./App.css"

function App() {

  const [vista, setVista] = useState("inicio")

  return (

    <div className="App">

      <nav className="navbar navbar-expand-lg navbar-modern">

        <div className="container-fluid">

          <div className="navbar-brand-container" onClick={() => setVista("inicio")}>

            <div className="brand-icon">

              <i className="fas fa-heartbeat"></i>

            </div>

            <span className="navbar-brand-text">Clínica Serenity</span>

          </div>

          <div className="navbar-nav-container">

            <button

              className={`nav-btn ${vista === "pacientes" ? "active" : ""}`}

              onClick={() => setVista("pacientes")}

            >

              <i className="fas fa-users"></i>

              <span>Pacientes</span>

            </button>

            <button className={`nav-btn ${vista === "medicos" ? "active" : ""}`} onClick={() => setVista("medicos")}>

              <i className="fas fa-user-md"></i>

              <span>Médicos</span>

            </button>

            <button className={`nav-btn ${vista === "citas" ? "active" : ""}`} onClick={() => setVista("citas")}>

              <i className="fas fa-calendar-alt"></i>

              <span>Citas</span>

            </button>

          </div>

        </div>

      </nav>

      <div className="main-content">

        {vista === "inicio" && <Inicio onIr={setVista} />}

        {vista === "pacientes" && <ListaPacientes />}

        {vista === "medicos" && <ListaMedicos />}

        {vista === "citas" && <ListaCitas />}

      </div>

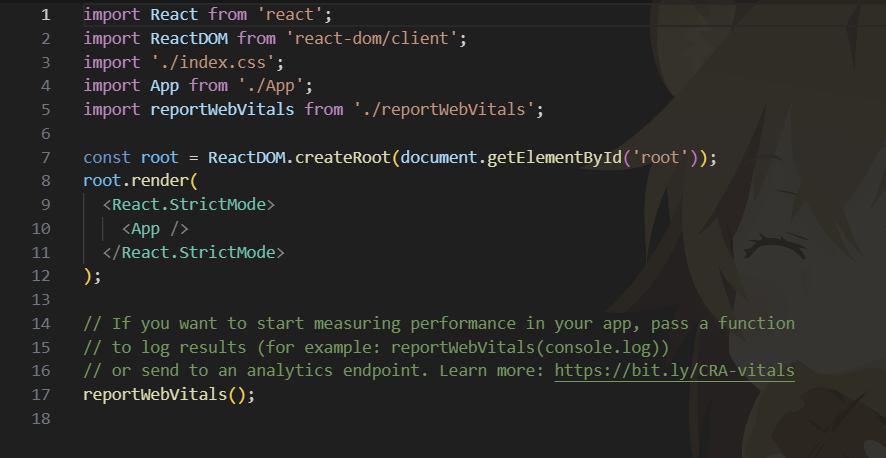
    </div>

  )

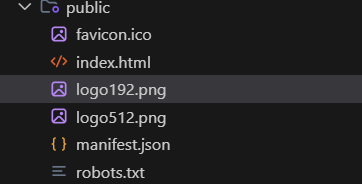
}

export default App

Index.js



* frontend/public/: Contiene los archivos estáticos como index.html por determinado.



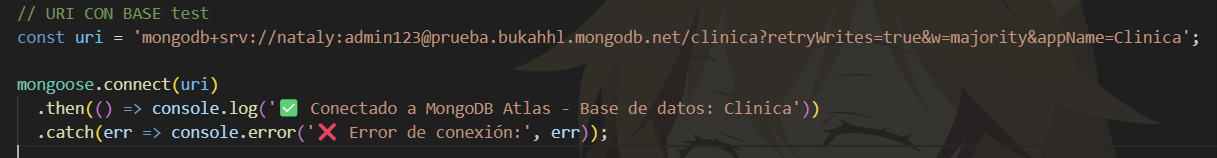




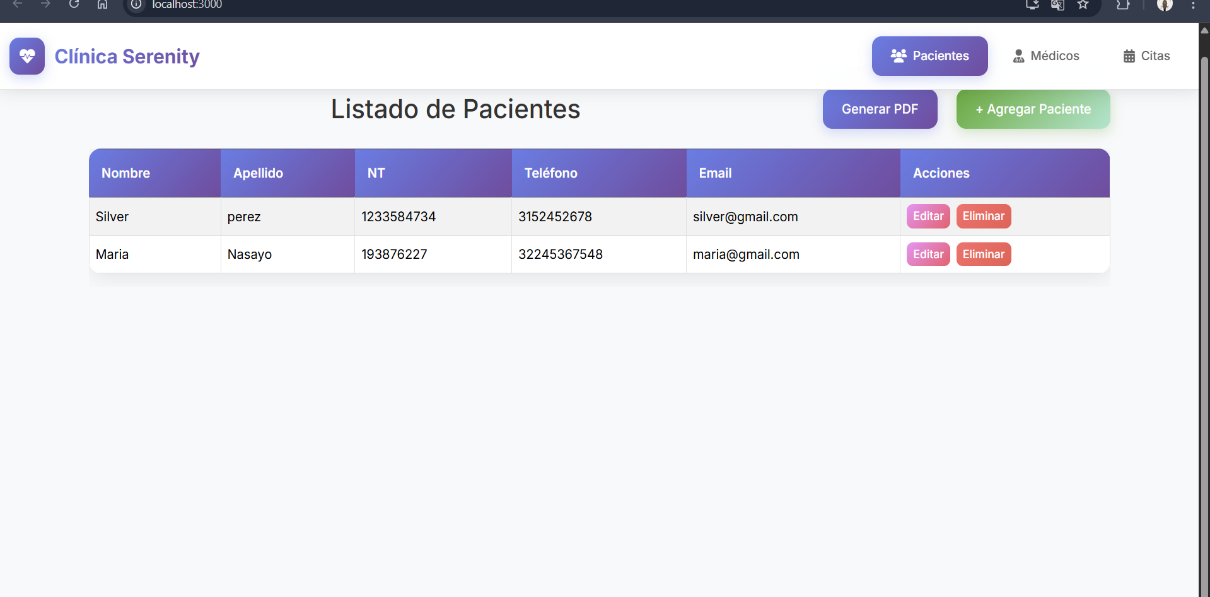
#### 3. Explicación de la Conexión con MongoDB

La aplicación "Clínica Serenity" utiliza MongoDB Atlas como su base de datos, lo que permite un almacenamiento de datos flexible y escalable en la nube. La conexión a la base de datos se gestiona en el backend a través de Mongoose, un modelado de objetos de MongoDB para Node.js.

El archivo backend/servees el encargado de establecer esta conexión/server.js



### Descripción del Frontend en React y Cómo se Conecta al Backend



El frontend de **Clínica Serenity** está desarrollado con **React**, una biblioteca de JavaScript diseñada para construir interfaces de usuario interactivas. La aplicación se enfoca en ofrecer una experiencia clara y funcional para la gestión de datos médicos.

#### 🧩 Componentes Principales:

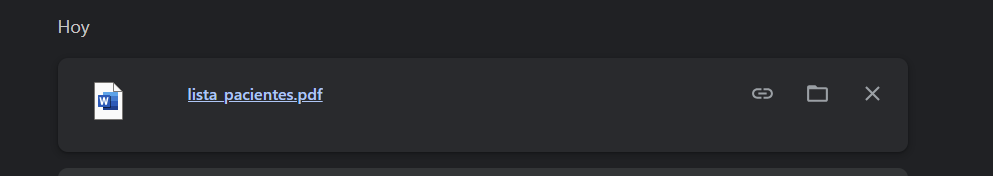
* **-Listas** (por ejemplo, ListaPacientes.jsx): Muestran los datos recuperados del backend en forma de tablas, incluyendo opciones para **editar**,**eliminar** y ahora también **generar PDFs**.
* **Formularios** (por ejemplo, PacienteForm.jsx): Permiten la **creación** y **edición** de registros de pacientes, médicos o citas.

#### 🧩Conexión con el Backend (Axios):

El frontend se comunica con el backend mediante solicitudes HTTP (GET, POST, PUT, DELETE) usando la librería **Axios**. Por ejemplo, para mostrar la lista de pacientes, se realiza una solicitud GET a la API del backend, que responde con los datos necesarios para llenar la tabla.

#### 🧩Generación de PDFs (Integración Frontend-Backend):

Una característica clave implementada es la capacidad de generar **reportes en PDF** directamente desde las listas.  
Cuando un usuario hace clic en el botón **"Generar PDF"**, el frontend realiza una solicitud GET a una ruta específica del backend (por ejemplo, /api/pacientes/pdf).  
El backend, utilizando la librería **Puppeteer**, genera el PDF a partir del HTML de los datos y lo envía al frontend como un archivo **blob binario**, que luego es descargado automáticamente por el navegador.



#### 5. Conclusiones Personales sobre lo Aprendido

Durante este proyecto, fortalecí mis conocimientos

* Comprendí cómo construir desde la base de datos hasta la interfaz del usuario, pasando por la lógica del servidor.
* Aprendí a crear rutas y controladores en Express, y a consumirlos desde React con Axios.
* Integré la generación de documentos, añadiendo funciones externas útiles en el PDF la cual fue una complicacion que el terminal ocupe:



-“npm install puppeteer”para automatizar la creación del PDF,

-“npm install jsonwebtoken”para gestionar la autenticación mediante tokens. Aunque fue complicado al principio,para lograrla conectar

En resumen, este proyecto me enseñó que aunque algunas partes sean complejas y tomen tiempo, con paciencia y esfuerzo se puede lograr y ademas lo mas lo generar permisos para descargar en pdf lo solucione con comandos fue tardado esa parte .

Estructura:Todo las carpetas dentro de Backend

**📁Backend/controladores**

Auth.js

const Usuario = require("../modelos/Usuario")

const Paciente = require("../modelos/Paciente")

const Medico = require("../modelos/Medico")

const jwt = require("jsonwebtoken")

// Clave secreta para JWT

const JWT\_SECRET = "clinica\_san\_jose\_secret\_key\_2025"

// Generar token JWT

const generarToken = (usuario) => {

  return jwt.sign(

    {

      id: usuario.\_id,

      email: usuario.email,

      tipo: usuario.tipo,

      entidadId: usuario.entidadId,

    },

    JWT\_SECRET,

    { expiresIn: "24h" },

  )

}

// Login

exports.login = async (req, res) => {

  try {

    const { email, password } = req.body

    // Verificar si el usuario existe

    const usuario = await Usuario.findOne({ email })

    if (!usuario) {

      return res.status(401).json({ error: "Credenciales inválidas" })

    }

    // Verificar la contraseña

    const passwordValido = await usuario.compararPassword(password)

    if (!passwordValido) {

      return res.status(401).json({ error: "Credenciales inválidas" })

    }

    // Generar token

    const token = generarToken(usuario)

    // Obtener datos adicionales según el tipo de usuario

    let datosAdicionales = null

    if (usuario.tipo === "medico" && usuario.entidadId) {

      datosAdicionales = await Medico.findById(usuario.entidadId)

    } else if (usuario.tipo === "paciente" && usuario.entidadId) {

      datosAdicionales = await Paciente.findById(usuario.entidadId)

    }

    // Responder con el token y datos del usuario

    res.json({

      token,

      usuario: {

        id: usuario.\_id,

        email: usuario.email,

        nombre: usuario.nombre,

        apellido: usuario.apellido,

        tipo: usuario.tipo,

        entidadId: usuario.entidadId,

        datosAdicionales,

      },

    })

  } catch (err) {

    res.status(500).json({ error: err.message })

  }

}

// Registro de paciente

exports.registrarPaciente = async (req, res) => {

  try {

    const { email, password, nombre, apellido, dni, telefono } = req.body

    // Verificar si el usuario ya existe

    const usuarioExistente = await Usuario.findOne({ email })

    if (usuarioExistente) {

      return res.status(400).json({ error: "El email ya está registrado" })

    }

    // Crear el paciente

    const nuevoPaciente = new Paciente({

      nombre,

      apellido,

      dni,

      telefono,

      email,

    })

    const pacienteGuardado = await nuevoPaciente.save()

    // Crear el usuario asociado al paciente

    const nuevoUsuario = new Usuario({

      email,

      password,

      nombre,

      apellido,

      tipo: "paciente",

      entidadId: pacienteGuardado.\_id,

    })

    await nuevoUsuario.save()

    // Generar token

    const token = generarToken(nuevoUsuario)

    res.status(201).json({

      token,

      usuario: {

        id: nuevoUsuario.\_id,

        email: nuevoUsuario.email,

        nombre: nuevoUsuario.nombre,

        apellido: nuevoUsuario.apellido,

        tipo: nuevoUsuario.tipo,

        entidadId: nuevoUsuario.entidadId,

        datosAdicionales: pacienteGuardado,

      },

    })

  } catch (err) {

    res.status(400).json({ error: err.message })

  }

}

// Registro de médico

exports.registrarMedico = async (req, res) => {

  try {

    const { email, password, nombre, apellido, especialidad, telefono, horario } = req.body

    // Verificar si el usuario ya existe

    const usuarioExistente = await Usuario.findOne({ email })

    if (usuarioExistente) {

      return res.status(400).json({ error: "El email ya está registrado" })

    }

    // Crear el médico

    const nuevoMedico = new Medico({

      nombre,

      apellido,

      especialidad,

      telefono,

      email,

      horario,

    })

    const medicoGuardado = await nuevoMedico.save()

    // Crear el usuario asociado al médico

    const nuevoUsuario = new Usuario({

      email,

      password,

      nombre,

      apellido,

      tipo: "medico",

      entidadId: medicoGuardado.\_id,

    })

    await nuevoUsuario.save()

    // Generar token

    const token = generarToken(nuevoUsuario)

    res.status(201).json({

      token,

      usuario: {

        id: nuevoUsuario.\_id,

        email: nuevoUsuario.email,

        nombre: nuevoUsuario.nombre,

        apellido: nuevoUsuario.apellido,

        tipo: nuevoUsuario.tipo,

        entidadId: nuevoUsuario.entidadId,

        datosAdicionales: medicoGuardado,

      },

    })

  } catch (err) {

    res.status(400).json({ error: err.message })

  }

}

// Registro de administrador

exports.registrarAdmin = async (req, res) => {

  try {

    const { email, password, nombre, apellido } = req.body

    // Verificar si el usuario ya existe

    const usuarioExistente = await Usuario.findOne({ email })

    if (usuarioExistente) {

      return res.status(400).json({ error: "El email ya está registrado" })

    }

    // Crear el usuario administrador

    const nuevoUsuario = new Usuario({

      email,

      password,

      nombre,

      apellido,

      tipo: "admin",

    })

    await nuevoUsuario.save()

    // Generar token

    const token = generarToken(nuevoUsuario)

    res.status(201).json({

      token,

      usuario: {

        id: nuevoUsuario.\_id,

        email: nuevoUsuario.email,

        nombre: nuevoUsuario.nombre,

        apellido: nuevoUsuario.apellido,

        tipo: nuevoUsuario.tipo,

      },

    })

  } catch (err) {

    res.status(400).json({ error: err.message })

  }

}

// Verificar token y obtener usuario actual

exports.verificarUsuario = async (req, res) => {

  try {

    const token = req.header("Authorization")?.replace("Bearer ", "")

    if (!token) {

      return res.status(401).json({ error: "Acceso denegado" })

    }

    try {

      const decoded = jwt.verify(token, JWT\_SECRET)

      const usuario = await Usuario.findById(decoded.id).select("-password")

      if (!usuario) {

        return res.status(404).json({ error: "Usuario no encontrado" })

      }

      // Obtener datos adicionales según el tipo de usuario

      let datosAdicionales = null

      if (usuario.tipo === "medico" && usuario.entidadId) {

        datosAdicionales = await Medico.findById(usuario.entidadId)

      } else if (usuario.tipo === "paciente" && usuario.entidadId) {

        datosAdicionales = await Paciente.findById(usuario.entidadId)

      }

      res.json({

        usuario: {

          id: usuario.\_id,

          email: usuario.email,

          nombre: usuario.nombre,

          apellido: usuario.apellido,

          tipo: usuario.tipo,

          entidadId: usuario.entidadId,

          datosAdicionales,

        },

      })

    } catch (error) {

      res.status(401).json({ error: "Token inválido" })

    }

  } catch (err) {

    res.status(500).json({ error: err.message })

  }

}

Cita.js

//controladores/cita.js

const Cita = require('../modelos/Cita');

exports.listarCitas = async (req, res) => {

  try {

    const citas = await Cita.find().populate('paciente').populate('medico');

    res.json(citas);

  } catch (err) {

    res.status(500).json({ error: err.message });

  }

};

exports.obtenerCita = async (req, res) => {

  try {

    const cita = await Cita.findById(req.params.id).populate('paciente').populate('medico');

    if (!cita) return res.status(404).json({ error: 'Cita no encontrada' });

    res.json(cita);

  } catch (err) {

    res.status(500).json({ error: err.message });

  }

};

exports.crearCita = async (req, res) => {

  try {

    const { medico, fecha, hora } = req.body;

    // Validar que no exista una cita igual para el mismo médico, fecha y hora

    const citaExistente = await Cita.findOne({ medico, fecha, hora });

    if (citaExistente) {

      return res.status(400).json({ error: 'El médico ya tiene una cita agendada en esa fecha y hora.' });

    }

    const nuevaCita = new Cita(req.body);

    await nuevaCita.save();

    res.status(201).json(nuevaCita);

  } catch (err) {

    res.status(400).json({ error: err.message });

  }

};

exports.actualizarCita = async (req, res) => {

  try {

    const { medico, fecha, hora } = req.body;

    // Validar que no exista otra cita igual para el mismo médico, fecha y hora (excluyendo la actual)

    const citaExistente = await Cita.findOne({ medico, fecha, hora, \_id: { $ne: req.params.id } });

    if (citaExistente) {

      return res.status(400).json({ error: 'El médico ya tiene una cita agendada en esa fecha y hora.' });

    }

    const cita = await Cita.findByIdAndUpdate(req.params.id, req.body, { new: true });

    if (!cita) return res.status(404).json({ error: 'Cita no encontrada' });

    res.json(cita);

  } catch (err) {

    res.status(400).json({ error: err.message });

  }

};

exports.eliminarCita = async (req, res) => {

  try {

    const cita = await Cita.findByIdAndDelete(req.params.id);

    if (!cita) return res.status(404).json({ error: 'Cita no encontrada' });

    res.json({ mensaje: 'Cita eliminada' });

  } catch (err) {

    res.status(500).json({ error: err.message });

  }

};

Medico.js

//controlladores/medico.js

const Medico = require('../modelos/Medico');

exports.listarMedicos = async (req, res) => {

  try {

    const medicos = await Medico.find();

    res.json(medicos);

  } catch (err) {

    res.status(500).json({ error: err.message });

  }

};

exports.obtenerMedico = async (req, res) => {

  try {

    const medico = await Medico.findById(req.params.id);

    if (!medico) return res.status(404).json({ error: 'Médico no encontrado' });

    res.json(medico);

  } catch (err) {

    res.status(500).json({ error: err.message });

  }

};

exports.crearMedico = async (req, res) => {

  try {

    const nuevoMedico = new Medico(req.body);

    await nuevoMedico.save();

    res.status(201).json(nuevoMedico);

  } catch (err) {

    res.status(400).json({ error: err.message });

  }

};

exports.actualizarMedico = async (req, res) => {

  try {

    const medico = await Medico.findByIdAndUpdate(req.params.id, req.body, { new: true });

    if (!medico) return res.status(404).json({ error: 'Médico no encontrado' });

    res.json(medico);

  } catch (err) {

    res.status(400).json({ error: err.message });

  }

};

exports.eliminarMedico = async (req, res) => {

  try {

    const medico = await Medico.findByIdAndDelete(req.params.id);

    if (!medico) return res.status(404).json({ error: 'Médico no encontrado' });

    res.json({ mensaje: 'Médico eliminado' });

  } catch (err) {

    res.status(500).json({ error: err.message });

  }

};

Paciente.js

// +//controladores/paciente.js

const Paciente = require('../modelos/Paciente');

exports.listarPacientes = async (req, res) => {

  try {

    const pacientes = await Paciente.find();

    res.json(pacientes);

  } catch (err) {

    res.status(500).json({ error: err.message });

  }

};

exports.obtenerPaciente = async (req, res) => {

  try {

    const paciente = await Paciente.findById(req.params.id);

    if (!paciente) return res.status(404).json({ error: 'Paciente no encontrado' });

    res.json(paciente);

  } catch (err) {

    res.status(500).json({ error: err.message });

  }

};

exports.crearPaciente = async (req, res) => {

  try {

    const nuevoPaciente = new Paciente(req.body);

    await nuevoPaciente.save();

    res.status(201).json(nuevoPaciente);

  } catch (err) {

    res.status(400).json({ error: err.message });

  }

};

exports.actualizarPaciente = async (req, res) => {

  try {

    const paciente = await Paciente.findByIdAndUpdate(req.params.id, req.body, { new: true });

    if (!paciente) return res.status(404).json({ error: 'Paciente no encontrado' });

    res.json(paciente);

  } catch (err) {

    res.status(400).json({ error: err.message });

  }

};

exports.eliminarPaciente = async (req, res) => {

  try {

    const paciente = await Paciente.findByIdAndDelete(req.params.id);

    if (!paciente) return res.status(404).json({ error: 'Paciente no encontrado' });

    res.json({ mensaje: 'Paciente eliminado' });

  } catch (err) {

    res.status(500).json({ error: err.message });

  }

};

Reportes.js

const Paciente = require("../modelos/Paciente")

const Medico = require("../modelos/Medico")

const Cita = require("../modelos/Cita")

const puppeteer = require('puppeteer');

exports.generarReporte = async (req, res) => {

  try {

    const { tipo, fechaInicio, fechaFin, usuarioId, tipoUsuario } = req.body

    let datos = {}

    let titulo = ""

    // Generar contenido según el tipo de reporte

    switch (tipo) {

      case "citas":

        datos = await generarReporteCitas(fechaInicio, fechaFin)

        titulo = "Reporte de Citas"

        break

      case "pacientes":

        datos = await generarReportePacientes()

        titulo = "Reporte de Pacientes"

        break

      case "medicos":

        datos = await generarReporteMedicos()

        titulo = "Reporte de Médicos"

        break

      case "mis-citas":

        datos = await generarReporteMisCitas(usuarioId, tipoUsuario, fechaInicio, fechaFin)

        titulo = "Mis Citas"

        break

      case "estadisticas":

        datos = await generarReporteEstadisticas()

        titulo = "Estadísticas Generales"

        break

      default:

        return res.status(400).json({ error: "Tipo de reporte no válido" })

    }

    // Generar HTML para el reporte

    const htmlReporte = generarHTMLReporte(titulo, datos, fechaInicio, fechaFin)

    res.json({

      titulo,

      html: htmlReporte,

      datos,

      fechaGeneracion: new Date().toLocaleDateString(),

    })

  } catch (error) {

    console.error("Error generando reporte:", error)

    const errorMessage = error.message || "Error al generar el reporte"

    res.status(500).json({ error: errorMessage })

  }

}

exports.generarPdfPacientes = async (req, res) => {

  let browser;

  try {

    const datos = await generarReportePacientes();

    const htmlReporte = generarHTMLReporte("Reporte de Pacientes", datos);

    browser = await puppeteer.launch({

      headless: true,

      args: ['--no-sandbox', '--disable-setuid-sandbox', '--disable-dev-shm-usage']

    });

    const page = await browser.newPage();

    await page.setContent(htmlReporte, { waitUntil: 'networkidle0' });

    const pdf = await page.pdf({ format: 'A4' });

    res.set({ 'Content-Type': 'application/pdf', 'Content-Length': pdf.length });

    res.send(pdf);

  } catch (error) {

    console.error("Error generando PDF de pacientes:", error);

    res.status(500).json({ error: "Error al generar el PDF de pacientes: " + error.message });

  } finally {

    if (browser) {

      await browser.close();

    }

  }

};

exports.generarPdfMedicos = async (req, res) => {

  let browser;

  try {

    const datos = await generarReporteMedicos();

    const htmlReporte = generarHTMLReporte("Reporte de Médicos", datos);

    browser = await puppeteer.launch({

      headless: true,

      args: ['--no-sandbox', '--disable-setuid-sandbox', '--disable-dev-shm-usage']

    });

    const page = await browser.newPage();

    await page.setContent(htmlReporte, { waitUntil: 'networkidle0' });

    const pdf = await page.pdf({ format: 'A4' });

    res.set({ 'Content-Type': 'application/pdf', 'Content-Length': pdf.length });

    res.send(pdf);

  } catch (error) {

    console.error("Error generando PDF de médicos:", error);

    res.status(500).json({ error: "Error al generar el PDF de médicos: " + error.message });

  } finally {

    if (browser) {

      await browser.close();

    }

  }

};

exports.generarPdfCitas = async (req, res) => {

  let browser;

  try {

    const datos = await generarReporteCitas();

    const htmlReporte = generarHTMLReporte("Reporte de Citas", datos);

    browser = await puppeteer.launch({

      headless: true,

      args: ['--no-sandbox', '--disable-setuid-sandbox', '--disable-dev-shm-usage']

    });

    const page = await browser.newPage();

    await page.setContent(htmlReporte, { waitUntil: 'networkidle0' });

    const pdf = await page.pdf({ format: 'A4' });

    res.set({ 'Content-Type': 'application/pdf', 'Content-Length': pdf.length });

    res.send(pdf);

  } catch (error) {

    console.error("Error generando PDF de citas:", error);

    res.status(500).json({ error: "Error al generar el PDF de citas: " + error.message });

  } finally {

    if (browser) {

      await browser.close();

    }

  }

};

async function generarReporteCitas(fechaInicio, fechaFin) {

  try {

    const query = {}

    if (fechaInicio && fechaFin) {

      query.fecha = {

        $gte: new Date(fechaInicio),

        $lte: new Date(fechaFin),

      }

    }

    const citas = await Cita.find(query)

      .populate("paciente", "nombre apellido dni")

      .populate("medico", "nombre apellido especialidad")

      .sort({ fecha: -1 })

    return {

      citas,

      total: citas.length,

      pendientes: citas.filter((c) => c.estado === "pendiente").length,

      confirmadas: citas.filter((c) => c.estado === "confirmada").length,

      canceladas: citas.filter((c) => c.estado === "cancelada").length,

    }

  } catch (error) {

    throw new Error("Error al generar reporte de citas")

  }

}

async function generarReportePacientes() {

  try {

    const pacientes = await Paciente.find().sort({ nombre: 1 })

    return {

      pacientes,

      total: pacientes.length,

    }

  } catch (error) {

    throw new Error("Error al generar reporte de pacientes")

  }

}

async function generarReporteMedicos() {

  try {

    const medicos = await Medico.find().sort({ nombre: 1 })

    return {

      medicos,

      total: medicos.length,

    }

  } catch (error) {

    throw new Error("Error al generar reporte de médicos")

  }

}

async function generarReporteMisCitas(usuarioId, tipoUsuario, fechaInicio, fechaFin) {

  try {

    const query = {}

    if (fechaInicio && fechaFin) {

      query.fecha = {

        $gte: new Date(fechaInicio),

        $lte: new Date(fechaFin),

      }

    }

    if (tipoUsuario === "medico") {

      query.medico = usuarioId

    } else if (tipoUsuario === "paciente") {

      query.paciente = usuarioId

    }

    const citas = await Cita.find(query)

      .populate("paciente", "nombre apellido dni")

      .populate("medico", "nombre apellido especialidad")

      .sort({ fecha: -1 })

    return {

      citas,

      total: citas.length,

      tipoUsuario,

    }

  } catch (error) {

    throw new Error("Error al generar reporte de citas")

  }

}

async function generarReporteEstadisticas() {

  try {

    const [totalPacientes, totalMedicos, totalCitas] = await Promise.all([

      Paciente.countDocuments(),

      Medico.countDocuments(),

      Cita.countDocuments(),

    ])

    const citasPorEstado = await Cita.aggregate([

      {

        $group: {

          \_id: "$estado",

          count: { $sum: 1 },

        },

      },

    ])

    const citasPorMes = await Cita.aggregate([

      {

        $group: {

          \_id: {

            year: { $year: "$fecha" },

            month: { $month: "$fecha" },

          },

          count: { $sum: 1 },

        },

      },

      { $sort: { "\_id.year": -1, "\_id.month": -1 } },

      { $limit: 12 },

    ])

    return {

      totalPacientes,

      totalMedicos,

      totalCitas,

      citasPorEstado,

      citasPorMes,

    }

  } catch (error) {

    throw new Error("Error al generar estadísticas")

  }

}

function generarHTMLReporte(titulo, datos, fechaInicio, fechaFin) {

  const fechaActual = new Date().toLocaleDateString()

  const periodo = fechaInicio && fechaFin ? `Del ${fechaInicio} al ${fechaFin}` : "Todos los registros"

  let contenido = ""

  if (datos.citas) {

    contenido = `

      <div class="seccion">

        <h3>Resumen</h3>

        <div class="stats-grid">

          <div class="stat-item">

            <span class="numero">${datos.total}</span>

            <span class="label">Total de Citas</span>

          </div>

          ${

            datos.pendientes !== undefined

              ? `

          <div class="stat-item">

            <span class="numero">${datos.pendientes}</span>

            <span class="label">Pendientes</span>

          </div>

          <div class="stat-item">

            <span class="numero">${datos.confirmadas}</span>

            <span class="label">Confirmadas</span>

          </div>

          <div class="stat-item">

            <span class="numero">${datos.canceladas}</span>

            <span class="label">Canceladas</span>

          </div>

          `

              : ""

          }

        </div>

      </div>

      <div class="seccion">

        <h3>Detalle de Citas</h3>

        <table class="tabla-reporte">

          <thead>

            <tr>

              <th>Fecha</th>

              <th>Hora</th>

              <th>Paciente</th>

              <th>Médico</th>

              <th>Estado</th>

              <th>Motivo</th>

            </tr>

          </thead>

          <tbody>

            ${datos.citas

              .map(

                (cita) => `

              <tr>

                <td>${new Date(cita.fecha).toLocaleDateString()}</td>

                <td>${cita.hora}</td>

                <td>${cita.paciente.nombre} ${cita.paciente.apellido}</td>

                <td>${cita.medico.nombre} ${cita.medico.apellido}<br><small>${cita.medico.especialidad}</small></td>

                <td><span class="estado estado-${cita.estado}">${cita.estado}</span></td>

                <td>${cita.motivo}</td>

              </tr>

            `,

              )

              .join("")}

          </tbody>

        </table>

      </div>

    `

  } else if (datos.pacientes) {

    contenido = `

      <div class="seccion">

        <h3>Resumen de Pacientes</h3>

        <div class="stats-grid">

          <div class="stat-item">

            <span class="numero">${datos.total}</span>

            <span class="label">Total de Pacientes</span>

          </div>

        </div>

      </div>

      <div class="seccion">

        <h3>Detalle de Pacientes</h3>

        <table class="tabla-reporte">

          <thead>

            <tr>

              <th>Nombre</th>

              <th>Apellido</th>

              <th>DNI</th>

              <th>Teléfono</th>

              <th>Email</th>

            </tr>

          </thead>

          <tbody>

            ${datos.pacientes

              .map(

                (paciente) => `

              <tr>

                <td>${paciente.nombre}</td>

                <td>${paciente.apellido}</td>

                <td>${paciente.dni}</td>

                <td>${paciente.telefono}</td>

                <td>${paciente.email}</td>

              </tr>

            `,

              )

              .join("")}

          </tbody>

        </table>

      </div>

    `

  } else if (datos.medicos) {

    contenido = `

      <div class="seccion">

        <h3>Resumen de Médicos</h3>

        <div class="stats-grid">

          <div class="stat-item">

            <span class="numero">${datos.total}</span>

            <span class="label">Total de Médicos</span>

          </div>

        </div>

      </div>

      <div class="seccion">

        <h3>Detalle de Médicos</h3>

        <table class="tabla-reporte">

          <thead>

            <tr>

              <th>Nombre</th>

              <th>Apellido</th>

              <th>Especialidad</th>

              <th>Teléfono</th>

              <th>Email</th>

            </tr>

          </thead>

          <tbody>

            ${datos.medicos

              .map(

                (medico) => `

              <tr>

                <td>${medico.nombre}</td>

                <td>${medico.apellido}</td>

                <td>${medico.especialidad}</td>

                <td>${medico.telefono}</td>

                <td>${medico.email}</td>

              </tr>

            `,

              )

              .join("")}

          </tbody>

        </table>

      </div>

    `

  } else if (datos.totalPacientes !== undefined) {

    contenido = `

      <div class="seccion">

        <h3>Estadísticas Generales</h3>

        <div class="stats-grid">

          <div class="stat-item">

            <span class="numero">${datos.totalPacientes}</span>

            <span class="label">Total Pacientes</span>

          </div>

          <div class="stat-item">

            <span class="numero">${datos.totalMedicos}</span>

            <span class="label">Total Médicos</span>

          </div>

          <div class="stat-item">

            <span class="numero">${datos.totalCitas}</span>

            <span class="label">Total Citas</span>

          </div>

        </div>

      </div>

      <div class="seccion">

        <h3>Citas por Estado</h3>

        <table class="tabla-reporte">

          <thead>

            <tr>

              <th>Estado</th>

              <th>Cantidad</th>

            </tr>

          </thead>

          <tbody>

            ${datos.citasPorEstado

              .map(

                (estado) => `

              <tr>

                <td><span class="estado estado-${estado.\_id}">${estado.\_id}</span></td>

                <td>${estado.count}</td>

              </tr>

            `,

              )

              .join("")}

          </tbody>

        </table>

      </div>

    `

  }

  return `

    <!DOCTYPE html>

    <html>

    <head>

      <meta charset="UTF-8">

      <title>${titulo}</title>

      <style>

        body {

          font-family: Arial, sans-serif;

          margin: 0;

          padding: 20px;

          background: white;

        }

        .header {

          text-align: center;

          margin-bottom: 30px;

          border-bottom: 2px solid #667eea;

          padding-bottom: 20px;

        }

        .header h1 {

          color: #667eea;

          margin: 0;

          font-size: 28px;

        }

        .header h2 {

          color: #333;

          margin: 10px 0;

          font-size: 20px;

        }

        .header p {

          color: #666;

          margin: 5px 0;

        }

        .seccion {

          margin-bottom: 30px;

        }

        .seccion h3 {

          color: #333;

          border-bottom: 1px solid #ddd;

          padding-bottom: 10px;

          margin-bottom: 20px;

        }

        .stats-grid {

          display: grid;

          grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));

          gap: 20px;

          margin-bottom: 30px;

        }

        .stat-item {

          text-align: center;

          padding: 20px;

          border: 1px solid #ddd;

          border-radius: 8px;

          background: #f8f9fa;

        }

        .stat-item .numero {

          display: block;

          font-size: 24px;

          font-weight: bold;

          color: #667eea;

          margin-bottom: 5px;

        }

        .stat-item .label {

          color: #666;

          font-size: 14px;

        }

        .tabla-reporte {

          width: 100%;

          border-collapse: collapse;

          margin-top: 20px;

        }

        .tabla-reporte th,

        .tabla-reporte td {

          border: 1px solid #ddd;

          padding: 12px;

          text-align: left;

        }

        .tabla-reporte th {

          background: #667eea;

          color: white;

          font-weight: bold;

        }

        .tabla-reporte tr:nth-child(even) {

          background: #f8f9fa;

        }

        .estado {

          padding: 4px 8px;

          border-radius: 4px;

          font-size: 12px;

          font-weight: bold;

          text-transform: uppercase;

        }

        .estado-pendiente {

          background: #fff3cd;

          color: #856404;

        }

        .estado-confirmada {

          background: #d4edda;

          color: #155724;

        }

        .estado-cancelada {

          background: #f8d7da;

          color: #721c24;

        }

        @media print {

          body { margin: 0; }

          .no-print { display: none; }

        }

      </style>

    </head>

    <body>

      <div class="header">

        <h1>Clínica Serenity</h1>

        <h2>${titulo}</h2>

        <p>Periodo: ${periodo}</p>

        <p>Fecha de generación: ${fechaActual}</p>

      </div>

      ${contenido}

      <div class="no-print" style="margin-top: 30px; text-align: center;">

        <button onclick="window.print()" style="background: #667eea; color: white; border: none; padding: 10px 20px; border-radius: 5px; cursor: pointer;">

          Imprimir / Guardar como PDF

        </button>

      </div>

    </body>

    </html>

  `

}

📁modelos

Cita.js

// File: backend/modelos/Cita.js

const mongoose = require('mongoose');

const CitaSchema = new mongoose.Schema({

  fecha: { type: Date, required: true },

  hora: { type: String, required: true },

  paciente: { type: mongoose.Schema.Types.ObjectId, ref: 'Paciente', required: true },

  medico: { type: mongoose.Schema.Types.ObjectId, ref: 'Medico', required: true },

  motivo: { type: String, required: true },

  estado: { type: String, enum: ['pendiente', 'confirmada', 'cancelada'], default: 'pendiente' }

});

module.exports = mongoose.model('Cita', CitaSchema);

Medico.js

// File: backend/modelos/Medico.js

const mongoose = require('mongoose');

const MedicoSchema = new mongoose.Schema({

  nombre: { type: String, required: true },

  apellido: { type: String, required: true },

  especialidad: { type: String, required: true },

  telefono: { type: String, required: true },

  email: { type: String, required: true, unique: true },

  horario: { type: String, required: true }

});

module.exports = mongoose.model('Medico', MedicoSchema);

Pasiente.js

// File: backend/modelos/Paciente.js

const mongoose = require('mongoose');

const PacienteSchema = new mongoose.Schema({

  nombre: { type: String, required: true },

  apellido: { type: String, required: true },

  dni: { type: String, required: true, unique: true },

  telefono: { type: String, required: true },

  email: { type: String, required: true, unique: true }

});

module.exports = mongoose.model('Paciente', PacienteSchema);

📁Rutas

Auth.js

const express = require("express")

const router = express.Router()

const authCtrl = require("../controladores/auth")

// Rutas de autenticación

router.post("/login", authCtrl.login)

router.post("/registro/paciente", authCtrl.registrarPaciente)

router.post("/registro/medico", authCtrl.registrarMedico)

router.post("/registro/admin", authCtrl.registrarAdmin)

router.get("/verificar", authCtrl.verificarUsuario)

module.exports = router

Citas.js

const express = require('express');

const router = express.Router();

const citaCtrl = require('../controladores/cita');

const reportesCtrl = require('../controladores/reportes');

const { verificarToken } = require('../middleware/auth');

router.get('/', citaCtrl.listarCitas);

router.get('/pdf', reportesCtrl.generarPdfCitas);

router.get('/:id', citaCtrl.obtenerCita);

router.post('/', citaCtrl.crearCita);

router.put('/:id', citaCtrl.actualizarCita);

router.delete('/:id', citaCtrl.eliminarCita);

module.exports = router;

Medico.js

const express = require('express');

const router = express.Router();

const medicoCtrl = require('../controladores/medico');

const reportesCtrl = require('../controladores/reportes');

const { verificarToken } = require('../middleware/auth');

router.get('/', medicoCtrl.listarMedicos);

router.get('/pdf', reportesCtrl.generarPdfMedicos);

router.get('/:id', medicoCtrl.obtenerMedico);

router.post('/', medicoCtrl.crearMedico);

router.put('/:id', medicoCtrl.actualizarMedico);

router.delete('/:id', medicoCtrl.eliminarMedico);

module.exports = router;

Paciente.js

const express = require('express');

const router = express.Router();

const pacienteCtrl = require('../controladores/paciente');

const reportesCtrl = require('../controladores/reportes');

const { verificarToken } = require('../middleware/auth');

router.get('/', pacienteCtrl.listarPacientes);

router.get('/pdf', reportesCtrl.generarPdfPacientes);

router.get('/:id', pacienteCtrl.obtenerPaciente);

router.post('/', pacienteCtrl.crearPaciente);

router.put('/:id', pacienteCtrl.actualizarPaciente);

router.delete('/:id', pacienteCtrl.eliminarPaciente);

module.exports = router;

Reportes.js

const express = require("express")

const router = express.Router()

const reportesCtrl = require("../controladores/reportes")

const { verificarToken } = require("../middleware/auth")

// Ruta para generar reportes

router.post("/generar", verificarToken, reportesCtrl.generarReporte)

module.exports = router

Estructura:Todo las clases dentro de frontend

**📁src/components**

///////////////////////////////////////////////////////////////////////////////////////////////////////////

* Propósito: Este es probablemente el componente que representa la página de inicio o el dashboard de la aplicación, el primer lugar al que llega un usuario después de iniciar sesión.
* Cómo funciona: Muestra una visión general de la información clave, enlaces a las diferentes secciones (pacientes, médicos, citas) o estadísticas resumidas.

**Inicio.jsx**

"use client"

import { useState, useEffect } from "react"

import "./Inicio.css"

function Inicio({ onIr }) {

  const [mousePosition, setMousePosition] = useState({ x: 0, y: 0 })

  const [isLoaded, setIsLoaded] = useState(false)

  const [currentStat, setCurrentStat] = useState(0)

  const stats = [

    { number: "24/7", label: "Disponible", icon: "🕐" },

    { number: "50000+", label: "Pacientes", icon: "👥" },

    { number: "50+", label: "Médicos", icon: "👨‍⚕️" },

    { number: "50000+", label: "Citas", icon: "📅" },

  ]

  useEffect(() => {

    setIsLoaded(true)

    // Animación de estadísticas rotativas

    const interval = setInterval(() => {

      setCurrentStat((prev) => (prev + 1) % 4) // Usar número fijo en lugar de stats.length

    }, 3000)

    return () => clearInterval(interval)

  }, []) // Dependencias vacías están bien aquí

  const handleMouseMove = (e) => {

    const rect = e.currentTarget.getBoundingClientRect()

    setMousePosition({

      x: ((e.clientX - rect.left) / rect.width) \* 100,

      y: ((e.clientY - rect.top) / rect.height) \* 100,

    })

  }

  return (

    <div className="inicio-container" onMouseMove={handleMouseMove}>

      {/\* Partículas flotantes de fondo \*/}

      <div className="particles-container">

        {[...Array(20)].map((\_, i) => (

          <div

            key={i}

            className="particle"

            style={{

              left: `${Math.random() \* 100}%`,

              animationDelay: `${Math.random() \* 5}s`,

              animationDuration: `${3 + Math.random() \* 4}s`,

            }}

          />

        ))}

      </div>

      {/\* Cursor personalizado que sigue el mouse \*/}

      <div

        className="custom-cursor"

        style={{

          left: `${mousePosition.x}%`,

          top: `${mousePosition.y}%`,

        }}

      />

      <div className={`hero-section ${isLoaded ? "loaded" : ""}`}>

        <div className="floating-elements">

          <div className="float-element float-1">💊</div>

          <div className="float-element float-2">🩺</div>

          <div className="float-element float-3">❤️</div>

          <div className="float-element float-4">🏥</div>

        </div>

        <div className="logo-container">

          <div className="logo-animated">

            <div className="logo-rings">

              <div className="ring ring-1"></div>

              <div className="ring ring-2"></div>

              <div className="ring ring-3"></div>

            </div>

            <div className="logo-icon">

              <div className="heartbeat">💗</div>

            </div>

          </div>

        </div>

        <h1 className="inicio-titulo">

          <span className="word" style={{ animationDelay: "0.1s" }}>

            Clínica

          </span>

          <span className="word" style={{ animationDelay: "0.3s" }}>

            Médica

          </span>

          <span className="word gradient-text" style={{ animationDelay: "0.5s" }}>

            Serenity

          </span>

        </h1>

        <div className="typing-container">

          <p className="typing-text"> Bienvenido a nuestro sistema :3 ❤️ </p>

        </div>

        {/\* Estadísticas dinámicas \*/}

        <div className="dynamic-stats">

          <div className="stat-display">

            <div className="stat-icon">{stats[currentStat].icon}</div>

            <div className="stat-content">

              <span className="stat-number">{stats[currentStat].number}</span>

              <span className="stat-label">{stats[currentStat].label}</span>

            </div>

          </div>

          <div className="stat-indicators">

            {stats.map((\_, index) => (

              <div key={index} className={`indicator ${index === currentStat ? "active" : ""}`} />

            ))}

          </div>

        </div>

      </div>

      <div className="features-section">

        <div className="section-title">

          <h2>Nuestros Servicios</h2>

          <div className="title-underline"></div>

        </div>

        <div className="features-grid">

          <div className="feature-card pacientes-card" onClick={() => onIr("pacientes")}>

            <div className="card-background"></div>

            <div className="card-glow"></div>

            <div className="card-content">

              <div className="card-icon">

                <div className="icon-container">

                  <i className="fas fa-users"></i>

                  <div className="icon-ripple"></div>

                </div>

              </div>

              <h3>Gestión de Pacientes</h3>

              <p>Sistema completo para el registro y seguimiento de pacientes</p>

              <div className="card-stats">

                <div className="stat-item">

                  <span className="number">50000+</span>

                  <span className="label">Registrados</span>

                </div>

              </div>

              <div className="hover-effect">

                <div className="ripple"></div>

              </div>

            </div>

          </div>

          <div className="feature-card medicos-card" onClick={() => onIr("medicos")}>

            <div className="card-background"></div>

            <div className="card-glow"></div>

            <div className="card-content">

              <div className="card-icon">

                <div className="icon-container">

                  <i className="fas fa-user-md"></i>

                  <div className="icon-ripple"></div>

                </div>

              </div>

              <h3>Equipo Médico</h3>

              <p>Gestión completa de médicos y especialidades</p>

              <div className="card-stats">

                <div className="stat-item">

                  <span className="number">50+</span>

                  <span className="label">Especialistas</span>

                </div>

              </div>

              <div className="hover-effect">

                <div className="ripple"></div>

              </div>

            </div>

          </div>

          <div className="feature-card citas-card" onClick={() => onIr("citas")}>

            <div className="card-background"></div>

            <div className="card-glow"></div>

            <div className="card-content">

              <div className="card-icon">

                <div className="icon-container">

                  <i className="fas fa-calendar-alt"></i>

                  <div className="icon-ripple"></div>

                </div>

              </div>

              <h3>Citas Médicas</h3>

              <p>Programación inteligente y gestión de citas</p>

              <div className="card-stats">

                <div className="stat-item">

                  <span className="number">50000+</span>

                  <span className="label">Programadas</span>

                </div>

              </div>

              <div className="hover-effect">

                <div className="ripple"></div>

              </div>

            </div>

          </div>

        </div>

      </div>

      {/\* Sección de contacto con efectos \*/}

      <div className="contact-section">

        <div className="contact-background">

          <div className="wave wave-1"></div>

          <div className="wave wave-2"></div>

          <div className="wave wave-3"></div>

        </div>

        <div className="contact-content">

          <h2 className="contact-title">Contáctanos</h2>

          <div className="contact-grid">

            <div className="contact-item" data-aos="fade-up" data-aos-delay="100">

              <div className="contact-icon">

                <i className="fas fa-map-marker-alt"></i>

                <div className="icon-bg"></div>

              </div>

              <div className="contact-info">

                <h4>Ubicación</h4>

                <p>

                  Calle Principal #123

                  <br />

                  Ciudad Salud

                </p>

              </div>

            </div>

            <div className="contact-item" data-aos="fade-up" data-aos-delay="200">

              <div className="contact-icon">

                <i className="fas fa-phone"></i>

                <div className="icon-bg"></div>

              </div>

              <div className="contact-info">

                <h4>Teléfono</h4>

                <p>(555) 123-4567</p>

              </div>

            </div>

            <div className="contact-item" data-aos="fade-up" data-aos-delay="400">

              <div className="contact-icon">

                <i className="fas fa-clock"></i>

                <div className="icon-bg"></div>

              </div>

              <div className="contact-info">

                <h4>Horarios</h4>

                <p>Lun - Vie: 8:00 AM - 6:00 PM</p>

              </div>

            </div>

          </div>

        </div>

      </div>

    </div>

  )

}

export default Inicio

**Inicio.css**

@import url("https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css");

@import url("https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700;800;900&display=swap");

\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

.inicio-container {

  font-family: "Inter", sans-serif;

  background: linear-gradient(135deg, #667eea 0%, #764ba2 50%, #f093fb 100%);

  min-height: 100vh;

  position: relative;

  overflow-x: hidden;

  cursor: none;

}

/\* Cursor personalizado \*/

.custom-cursor {

  position: absolute;

  width: 20px;

  height: 20px;

  background: radial-gradient(circle, rgba(255, 255, 255, 0.8) 0%, rgba(255, 255, 255, 0.2) 70%, transparent 100%);

  border-radius: 50%;

  pointer-events: none;

  z-index: 9999;

  transition: all 0.1s ease;

  mix-blend-mode: difference;

}

/\* Partículas flotantes \*/

.particles-container {

  position: absolute;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  overflow: hidden;

  z-index: 1;

}

.particle {

  position: absolute;

  width: 4px;

  height: 4px;

  background: rgba(255, 255, 255, 0.6);

  border-radius: 50%;

  animation: float infinite ease-in-out;

}

@keyframes float {

  0%,

  100% {

    transform: translateY(0px) rotate(0deg);

    opacity: 1;

  }

  50% {

    transform: translateY(-100px) rotate(180deg);

    opacity: 0.5;

  }

}

/\* Hero Section \*/

.hero-section {

  position: relative;

  text-align: center;

  padding: 4rem 2rem;

  min-height: 100vh;

  display: flex;

  flex-direction: column;

  justify-content: center;

  align-items: center;

  z-index: 2;

  opacity: 0;

  transform: translateY(50px);

  transition: all 1s cubic-bezier(0.25, 0.46, 0.45, 0.94);

}

.hero-section.loaded {

  opacity: 1;

  transform: translateY(0);

}

/\* Elementos flotantes \*/

.floating-elements {

  position: absolute;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  pointer-events: none;

}

.float-element {

  position: absolute;

  font-size: 2rem;

  opacity: 0.3;

  animation: floatAround 8s infinite ease-in-out;

}

.float-1 {

  top: 20%;

  left: 10%;

  animation-delay: 0s;

}

.float-2 {

  top: 30%;

  right: 15%;

  animation-delay: 2s;

}

.float-3 {

  bottom: 30%;

  left: 20%;

  animation-delay: 4s;

}

.float-4 {

  bottom: 20%;

  right: 10%;

  animation-delay: 6s;

}

@keyframes floatAround {

  0%,

  100% {

    transform: translate(0, 0) rotate(0deg);

  }

  25% {

    transform: translate(20px, -20px) rotate(90deg);

  }

  50% {

    transform: translate(-10px, -30px) rotate(180deg);

  }

  75% {

    transform: translate(-20px, 10px) rotate(270deg);

  }

}

/\* Logo animado \*/

.logo-container {

  position: relative;

  margin-bottom: 3rem;

}

.logo-animated {

  position: relative;

  width: 150px;

  height: 150px;

  margin: 0 auto;

}

.logo-rings {

  position: absolute;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

}

.ring {

  position: absolute;

  border: 2px solid rgba(255, 255, 255, 0.3);

  border-radius: 50%;

  animation: pulse-ring 2s infinite;

}

.ring-1 {

  width: 100%;

  height: 100%;

  animation-delay: 0s;

}

.ring-2 {

  width: 80%;

  height: 80%;

  top: 10%;

  left: 10%;

  animation-delay: 0.5s;

}

.ring-3 {

  width: 60%;

  height: 60%;

  top: 20%;

  left: 20%;

  animation-delay: 1s;

}

@keyframes pulse-ring {

  0% {

    transform: scale(1);

    opacity: 1;

  }

  50% {

    transform: scale(1.1);

    opacity: 0.7;

  }

  100% {

    transform: scale(1.2);

    opacity: 0;

  }

}

.logo-icon {

  position: relative;

  z-index: 3;

  width: 100%;

  height: 100%;

  display: flex;

  align-items: center;

  justify-content: center;

  background: rgba(255, 255, 255, 0.1);

  border-radius: 50%;

  backdrop-filter: blur(20px);

  border: 2px solid rgba(255, 255, 255, 0.2);

}

.heartbeat {

  font-size: 4rem;

  animation: heartbeat 1.5s infinite;

}

@keyframes heartbeat {

  0%,

  50%,

  100% {

    transform: scale(1);

  }

  25%,

  75% {

    transform: scale(1.1);

  }

}

/\* Título animado \*/

.inicio-titulo {

  font-size: 4rem;

  font-weight: 900;

  margin-bottom: 2rem;

  color: white;

  text-shadow: 0 4px 20px rgba(0, 0, 0, 0.3);

}

.word {

  display: inline-block;

  opacity: 0;

  transform: translateY(50px);

  animation: slideInUp 0.8s forwards;

  margin-right: 0.5rem;

}

.gradient-text {

  background: linear-gradient(45deg, #ff6b6b, #4ecdc4, #45b7d1, #96ceb4);

  background-size: 300% 300%;

  -webkit-background-clip: text;

  -webkit-text-fill-color: transparent;

  background-clip: text;

  animation: gradientShift 3s ease-in-out infinite, slideInUp 0.8s forwards;

}

@keyframes slideInUp {

  to {

    opacity: 1;

    transform: translateY(0);

  }

}

@keyframes gradientShift {

  0%,

  100% {

    background-position: 0% 50%;

  }

  50% {

    background-position: 100% 50%;

  }

}

/\* Efecto de escritura \*/

.typing-container {

  margin-bottom: 3rem;

}

.typing-text {

  font-size: 1.5rem;

  color: rgba(255, 255, 255, 0.9);

  border-right: 2px solid white;

  white-space: nowrap;

  overflow: hidden;

  animation: typing 3s steps(40, end), blink-caret 0.75s step-end infinite;

}

@keyframes typing {

  from {

    width: 0;

  }

  to {

    width: 100%;

  }

}

@keyframes blink-caret {

  from,

  to {

    border-color: transparent;

  }

  50% {

    border-color: white;

  }

}

/\* Estadísticas dinámicas \*/

.dynamic-stats {

  display: flex;

  flex-direction: column;

  align-items: center;

  gap: 1rem;

}

.stat-display {

  display: flex;

  align-items: center;

  gap: 1rem;

  padding: 1.5rem 2rem;

  background: rgba(255, 255, 255, 0.1);

  border-radius: 20px;

  backdrop-filter: blur(20px);

  border: 1px solid rgba(255, 255, 255, 0.2);

  transition: all 0.5s ease;

  animation: statSlide 0.5s ease;

}

@keyframes statSlide {

  from {

    opacity: 0;

    transform: translateX(-50px);

  }

  to {

    opacity: 1;

    transform: translateX(0);

  }

}

.stat-icon {

  font-size: 2.5rem;

  animation: bounce 2s infinite;

}

@keyframes bounce {

  0%,

  20%,

  50%,

  80%,

  100% {

    transform: translateY(0);

  }

  40% {

    transform: translateY(-10px);

  }

  60% {

    transform: translateY(-5px);

  }

}

.stat-content {

  display: flex;

  flex-direction: column;

  color: white;

}

.stat-number {

  font-size: 2rem;

  font-weight: 800;

  line-height: 1;

}

.stat-label {

  font-size: 1rem;

  opacity: 0.8;

}

.stat-indicators {

  display: flex;

  gap: 0.5rem;

}

.indicator {

  width: 8px;

  height: 8px;

  border-radius: 50%;

  background: rgba(255, 255, 255, 0.3);

  transition: all 0.3s ease;

}

.indicator.active {

  background: white;

  transform: scale(1.2);

}

/\* Sección de características \*/

.features-section {

  padding: 6rem 2rem;

  background: rgba(255, 255, 255, 0.05);

  backdrop-filter: blur(10px);

}

.section-title {

  text-align: center;

  margin-bottom: 4rem;

}

.section-title h2 {

  font-size: 3rem;

  font-weight: 800;

  color: white;

  margin-bottom: 1rem;

}

.title-underline {

  width: 100px;

  height: 4px;

  background: linear-gradient(45deg, #ff6b6b, #4ecdc4);

  margin: 0 auto;

  border-radius: 2px;

  animation: expandLine 1s ease-out;

}

@keyframes expandLine {

  from {

    width: 0;

  }

  to {

    width: 100px;

  }

}

.features-grid {

  display: grid;

  grid-template-columns: repeat(auto-fit, minmax(350px, 1fr));

  gap: 3rem;

  max-width: 1200px;

  margin: 0 auto;

}

/\* Tarjetas de características mejoradas \*/

.feature-card {

  position: relative;

  background: rgba(255, 255, 255, 0.1);

  border-radius: 25px;

  padding: 3rem;

  cursor: pointer;

  transition: all 0.5s cubic-bezier(0.175, 0.885, 0.32, 1.275);

  backdrop-filter: blur(20px);

  border: 1px solid rgba(255, 255, 255, 0.2);

  overflow: hidden;

  transform: translateY(0);

}

.card-background {

  position: absolute;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  opacity: 0;

  transition: opacity 0.5s ease;

}

.pacientes-card .card-background {

  background: linear-gradient(135deg, #667eea, #764ba2);

}

.medicos-card .card-background {

  background: linear-gradient(135deg, #f093fb, #f5576c);

}

.citas-card .card-background {

  background: linear-gradient(135deg, #4facfe, #00f2fe);

}

.card-glow {

  position: absolute;

  top: -50%;

  left: -50%;

  width: 200%;

  height: 200%;

  background: radial-gradient(circle, rgba(255, 255, 255, 0.1) 0%, transparent 70%);

  opacity: 0;

  transition: opacity 0.5s ease;

}

.feature-card:hover {

  transform: translateY(-20px) scale(1.05);

  box-shadow: 0 30px 60px rgba(0, 0, 0, 0.3);

}

.feature-card:hover .card-background {

  opacity: 1;

}

.feature-card:hover .card-glow {

  opacity: 1;

}

.card-content {

  position: relative;

  z-index: 2;

  color: white;

}

.card-icon {

  margin-bottom: 2rem;

}

.icon-container {

  position: relative;

  width: 80px;

  height: 80px;

  margin: 0 auto;

  display: flex;

  align-items: center;

  justify-content: center;

  background: rgba(255, 255, 255, 0.2);

  border-radius: 20px;

  font-size: 2rem;

  transition: all 0.3s ease;

}

.icon-ripple {

  position: absolute;

  top: 50%;

  left: 50%;

  width: 0;

  height: 0;

  border-radius: 50%;

  background: rgba(255, 255, 255, 0.3);

  transform: translate(-50%, -50%);

  transition: all 0.5s ease;

}

.feature-card:hover .icon-container {

  transform: scale(1.1) rotate(5deg);

}

.feature-card:hover .icon-ripple {

  width: 120px;

  height: 120px;

}

.card-content h3 {

  font-size: 1.8rem;

  font-weight: 700;

  margin-bottom: 1rem;

  text-align: center;

}

.card-content p {

  font-size: 1.1rem;

  line-height: 1.6;

  opacity: 0.9;

  margin-bottom: 2rem;

  text-align: center;

}

.card-stats {

  display: flex;

  justify-content: center;

  margin-top: 1.5rem;

}

.stat-item {

  text-align: center;

  padding: 1rem;

  background: rgba(255, 255, 255, 0.1);

  border-radius: 15px;

  backdrop-filter: blur(10px);

}

.stat-item .number {

  display: block;

  font-size: 1.5rem;

  font-weight: 700;

}

.stat-item .label {

  font-size: 0.9rem;

  opacity: 0.8;

}

.hover-effect {

  position: absolute;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  pointer-events: none;

}

.ripple {

  position: absolute;

  border-radius: 50%;

  background: rgba(255, 255, 255, 0.3);

  transform: scale(0);

  animation: ripple-effect 0.6s linear;

}

@keyframes ripple-effect {

  to {

    transform: scale(4);

    opacity: 0;

  }

}

/\* Sección de contacto \*/

.contact-section {

  position: relative;

  padding: 6rem 2rem;

  background: linear-gradient(135deg, #667eea, #764ba2);

  overflow: hidden;

}

.contact-background {

  position: absolute;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

}

.wave {

  position: absolute;

  bottom: 0;

  left: 0;

  width: 100%;

  height: 100px;

  background: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 1200 120'%3E%3Cpath d='M0,0V46.29c47.79,22.2,103.59,32.17,158,28,70.36-5.37,136.33-33.31,206.8-37.5C438.64,32.43,512.34,53.67,583,72.05c69.27,18,138.3,24.88,209.4,13.08,36.15-6,69.85-17.84,104.45-29.34C989.49,25,1113-14.29,1200,52.47V0Z' opacity='.25' fill='%23FFFFFF'%3E%3C/path%3E%3C/svg%3E")

    repeat-x;

  animation: wave 10s ease-in-out infinite;

}

.wave-2 {

  animation-delay: -2s;

  opacity: 0.5;

}

.wave-3 {

  animation-delay: -4s;

  opacity: 0.3;

}

@keyframes wave {

  0%,

  100% {

    transform: translateX(0);

  }

  50% {

    transform: translateX(-25%);

  }

}

.contact-content {

  position: relative;

  z-index: 2;

  max-width: 1000px;

  margin: 0 auto;

}

.contact-title {

  text-align: center;

  font-size: 3rem;

  font-weight: 800;

  color: white;

  margin-bottom: 4rem;

  text-shadow: 0 4px 20px rgba(0, 0, 0, 0.3);

}

.contact-grid {

  display: grid;

  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

  gap: 2rem;

}

.contact-item {

  display: flex;

  align-items: center;

  gap: 1.5rem;

  padding: 2rem;

  background: rgba(255, 255, 255, 0.1);

  border-radius: 20px;

  backdrop-filter: blur(20px);

  border: 1px solid rgba(255, 255, 255, 0.2);

  transition: all 0.5s ease;

  transform: translateY(50px);

  opacity: 0;

  animation: slideUp 0.8s forwards;

}

.contact-item:nth-child(1) {

  animation-delay: 0.1s;

}

.contact-item:nth-child(2) {

  animation-delay: 0.2s;

}

.contact-item:nth-child(3) {

  animation-delay: 0.3s;

}

.contact-item:nth-child(4) {

  animation-delay: 0.4s;

}

@keyframes slideUp {

  to {

    transform: translateY(0);

    opacity: 1;

  }

}

.contact-item:hover {

  transform: translateY(-10px) scale(1.05);

  box-shadow: 0 20px 40px rgba(0, 0, 0, 0.2);

}

.contact-icon {

  position: relative;

  width: 60px;

  height: 60px;

  display: flex;

  align-items: center;

  justify-content: center;

  font-size: 1.5rem;

  color: white;

  border-radius: 15px;

}

.icon-bg {

  position: absolute;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  background: linear-gradient(45deg, rgba(255, 255, 255, 0.2), rgba(255, 255, 255, 0.1));

  border-radius: 15px;

  animation: iconGlow 2s ease-in-out infinite alternate;

}

@keyframes iconGlow {

  0% {

    box-shadow: 0 0 5px rgba(255, 255, 255, 0.3);

  }

  100% {

    box-shadow: 0 0 20px rgba(255, 255, 255, 0.6);

  }

}

.contact-info {

  color: white;

}

.contact-info h4 {

  font-size: 1.2rem;

  font-weight: 600;

  margin-bottom: 0.5rem;

}

.contact-info p {

  font-size: 1rem;

  opacity: 0.9;

  line-height: 1.4;

}

/\* Responsive \*/

@media (max-width: 768px) {

  .inicio-titulo {

    font-size: 2.5rem;

  }

  .features-grid {

    grid-template-columns: 1fr;

    gap: 2rem;

  }

  .feature-card {

    padding: 2rem;

  }

  .contact-grid {

    grid-template-columns: 1fr;

  }

  .hero-section {

    padding: 2rem 1rem;

  }

  .section-title h2 {

    font-size: 2rem;

  }

  .contact-title {

    font-size: 2rem;

  }

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

* Propósito: Estos componentes son responsables de mostrar listados de datos de una entidad específica (pacientes, médicos, citas).
* Cómo funcionan: Realizan una solicitud al backend para obtener todos los registros de esa entidad y luego los presentan en una tabla o lista. También suelen incluir botones o funcionalidades para:

-Editar un registro individual (que probablemente abre el \*Form.jsx correspondiente).

-Eliminar un registro.

-Generar PDF de la lista (funcionalidad que acabamos de implementar).

**Listacitas.jsx**

import React, { useEffect, useState } from 'react';

import axios from 'axios';

import 'bootstrap/dist/css/bootstrap.min.css';

import CitaForm from './CitaForm';

function ListaCitas() {

  const [citas, setCitas] = useState([]);

  const [error, setError] = useState(null);

  const [loading, setLoading] = useState(true);

  const [showForm, setShowForm] = useState(false);

  const [citaEditar, setCitaEditar] = useState(null);

  const cargarCitas = () => {

    setLoading(true);

    axios.get('http://localhost:4000/api/citas')

      .then(res => {

        setCitas(res.data);

        setLoading(false);

      })

      .catch(err => {

        setError('No se pudo cargar la lista de citas');

        setLoading(false);

      });

  };

  useEffect(() => {

    cargarCitas();

  }, []);

  const handleAgregar = () => {

    setCitaEditar(null);

    setShowForm(true);

  };

  const handleEditar = cita => {

    setCitaEditar(cita);

    setShowForm(true);

  };

  const handleEliminar = async id => {

    if (window.confirm('¿Seguro que deseas eliminar esta cita?')) {

      try {

        await axios.delete(`http://localhost:4000/api/citas/${id}`);

        cargarCitas();

      } catch (err) {

        setError('No se pudo eliminar la cita');

      }

    }

  };

  const handleFormSuccess = () => {

    setShowForm(false);

    cargarCitas();

  };

  const handleGenerarPdf = async () => {

    try {

      const response = await axios.get('http://localhost:4000/api/citas/pdf', { responseType: 'blob' });

      const blob = new Blob([response.data], { type: 'application/pdf' });

      const url = window.URL.createObjectURL(blob);

      const a = document.createElement('a');

      a.href = url;

      a.download = 'citas\_medicas.pdf';

      document.body.appendChild(a);

      a.click();

      document.body.removeChild(a);

      window.URL.revokeObjectURL(url);

      setError(null);

    } catch (err) {

      setError(err.response?.data?.error || 'No se pudo generar el PDF.');

    }

  };

  return (

    <div className="container mt-4">

      <h2 className="text-center mb-4">Listado de Citas Médicas</h2>

      <div className="mb-3 text-end">

        <button className="btn btn-primary me-2" onClick={handleGenerarPdf}>Generar PDF</button>

        <button className="btn btn-primary" onClick={handleAgregar}>Agendar Cita</button>

      </div>

      {showForm && (

        <div className="mb-4">

          <CitaForm

            cita={citaEditar}

            onSuccess={handleFormSuccess}

            onCancel={() => setShowForm(false)}

          />

        </div>

      )}

      {loading && <div className="alert alert-info">Cargando...</div>}

      {error && <div className="alert alert-danger">{error}</div>}

      <div className="table-responsive">

        <table className="table table-striped table-bordered align-middle">

          <thead className="table-dark">

            <tr>

              <th>Fecha</th>

              <th>Hora</th>

              <th>Paciente</th>

              <th>Médico</th>

              <th>Motivo</th>

              <th>Estado</th>

              <th>Acciones</th>

            </tr>

          </thead>

          <tbody>

            {citas.length === 0 && !loading ? (

              <tr>

                <td colSpan="7" className="text-center">No hay citas programadas</td>

              </tr>

            ) : (

              citas.map(cita => (

                <tr key={cita.\_id}>

                  <td>{cita.fecha ? cita.fecha.substring(0, 10) : ''}</td>

                  <td>{cita.hora}</td>

                  <td>{cita.paciente?.nombre} {cita.paciente?.apellido}</td>

                  <td>{cita.medico?.nombre} {cita.medico?.apellido} ({cita.medico?.especialidad})</td>

                  <td>{cita.motivo}</td>

                  <td>{cita.estado}</td>

                  <td>

                    <button className="btn btn-warning btn-sm me-2" onClick={() => handleEditar(cita)}>Editar</button>

                    <button className="btn btn-danger btn-sm" onClick={() => handleEliminar(cita.\_id)}>Eliminar</button>

                  </td>

                </tr>

              ))

            )}

          </tbody>

        </table>

      </div>

    </div>

  );

}

export default ListaCitas;

**Listamedico.jsx**

import React, { useEffect, useState } from 'react';

import axios from 'axios';

import 'bootstrap/dist/css/bootstrap.min.css';

import MedicoForm from './MedicoForm';

function ListaMedicos() {

  const [medicos, setMedicos] = useState([]);

  const [error, setError] = useState(null);

  const [loading, setLoading] = useState(true);

  const [showForm, setShowForm] = useState(false);

  const [medicoEditar, setMedicoEditar] = useState(null);

  const cargarMedicos = () => {

    setLoading(true);

    axios.get('http://localhost:4000/api/medicos')

      .then(res => {

        setMedicos(res.data);

        setLoading(false);

      })

      .catch(err => {

        setError('No se pudo cargar la lista de médicos');

        setLoading(false);

      });

  };

  useEffect(() => {

    cargarMedicos();

  }, []);

  const handleAgregar = () => {

    setMedicoEditar(null);

    setShowForm(true);

  };

  const handleEditar = medico => {

    setMedicoEditar(medico);

    setShowForm(true);

  };

  const handleEliminar = async id => {

    if (window.confirm('¿Seguro que deseas eliminar este médico?')) {

      try {

        await axios.delete(`http://localhost:4000/api/medicos/${id}`);

        cargarMedicos();

      } catch (err) {

        setError('No se pudo eliminar el médico');

      }

    }

  };

  const handleFormSuccess = () => {

    setShowForm(false);

    cargarMedicos();

  };

  const handleGenerarPdf = async () => {

    try {

      const response = await axios.get('http://localhost:4000/api/medicos/pdf', { responseType: 'blob' });

      const blob = new Blob([response.data], { type: 'application/pdf' });

      const url = window.URL.createObjectURL(blob);

      const a = document.createElement('a');

      a.href = url;

      a.download = 'lista\_medicos.pdf';

      document.body.appendChild(a);

      a.click();

      document.body.removeChild(a);

      window.URL.revokeObjectURL(url);

      setError(null);

    } catch (err) {

      setError(err.response?.data?.error || 'No se pudo generar el PDF.');

    }

  };

  return (

    <div className="container mt-4">

      <h2 className="text-center mb-4">Listado de Médicos</h2>

      <div className="mb-3 text-end">

        <button className="btn btn-primary me-2" onClick={handleGenerarPdf}>Generar PDF</button>

        <button className="btn btn-primary" onClick={handleAgregar}>Agregar Médico</button>

      </div>

      {showForm && (

        <div className="mb-4">

          <MedicoForm

            medico={medicoEditar}

            onSuccess={handleFormSuccess}

            onCancel={() => setShowForm(false)}

          />

        </div>

      )}

      {loading && <div className="alert alert-info">Cargando...</div>}

      {error && <div className="alert alert-danger">{error}</div>}

      <div className="table-responsive">

        <table className="table table-striped table-bordered align-middle">

          <thead className="table-dark">

            <tr>

              <th>Nombre</th>

              <th>Apellido</th>

              <th>Especialidad</th>

              <th>Teléfono</th>

              <th>Email</th>

              <th>Horario</th>

              <th>Acciones</th>

            </tr>

          </thead>

          <tbody>

            {medicos.length === 0 && !loading ? (

              <tr>

                <td colSpan="7" className="text-center">No hay médicos registrados</td>

              </tr>

            ) : (

              medicos.map(medico => (

                <tr key={medico.\_id}>

                  <td>{medico.nombre}</td>

                  <td>{medico.apellido}</td>

                  <td>{medico.especialidad}</td>

                  <td>{medico.telefono}</td>

                  <td>{medico.email}</td>

                  <td>{medico.horario}</td>

                  <td>

                    <button className="btn btn-warning btn-sm me-2" onClick={() => handleEditar(medico)}>Editar</button>

                    <button className="btn btn-danger btn-sm" onClick={() => handleEliminar(medico.\_id)}>Eliminar</button>

                  </td>

                </tr>

              ))

            )}

          </tbody>

        </table>

      </div>

    </div>

  );

}

export default ListaMedicos;

**Listapacientes.jsx**

import React, { useEffect, useState } from 'react';

import axios from 'axios';

import 'bootstrap/dist/css/bootstrap.min.css';

import PacienteForm from './PacienteForm';

function ListaPacientes() {

  const [pacientes, setPacientes] = useState([]);

  const [error, setError] = useState(null);

  const [loading, setLoading] = useState(true);

  const [showForm, setShowForm] = useState(false);

  const [pacienteEditar, setPacienteEditar] = useState(null);

  const cargarPacientes = () => {

    setLoading(true);

    axios.get('http://localhost:4000/api/pacientes')

      .then(res => {

        setPacientes(res.data);

        setLoading(false);

      })

      .catch(err => {

        setError('No se pudo cargar la lista de pacientes');

        setLoading(false);

      });

  };

  useEffect(() => {

    cargarPacientes();

  }, []);

  const handleAgregar = () => {

    setPacienteEditar(null);

    setShowForm(true);

  };

  const handleEditar = paciente => {

    setPacienteEditar(paciente);

    setShowForm(true);

  };

  const handleEliminar = async id => {

    if (window.confirm('¿Seguro que deseas eliminar este paciente?')) {

      try {

        await axios.delete(`http://localhost:4000/api/pacientes/${id}`);

        cargarPacientes();

      } catch (err) {

        setError('No se pudo eliminar el paciente');

      }

    }

  };

  const handleFormSuccess = () => {

    setShowForm(false);

    cargarPacientes();

  };

  const handleGenerarPdf = async () => {

    try {

      const response = await axios.get('http://localhost:4000/api/pacientes/pdf', { responseType: 'blob' });

      const url = window.URL.createObjectURL(new Blob([response.data]));

      const link = document.createElement('a');

      link.href = url;

      link.setAttribute('download', 'lista\_pacientes.pdf');

      document.body.appendChild(link);

      link.click();

      link.remove();

      setError(null);

    } catch (err) {

      setError(err.response?.data?.error || 'No se pudo generar el PDF.');

    }

  };

  return (

    <div className="container mt-4">

      <div className="d-flex justify-content-between align-items-center mb-4">

        <h2 className="text-center flex-grow-1 mb-0">Listado de Pacientes</h2>

        <button className="btn btn-primary me-2" onClick={handleGenerarPdf}>Generar PDF</button>

        <button className="btn btn-success ms-3" onClick={handleAgregar}>+ Agregar Paciente</button>

      </div>

      {showForm && (

        <div className="mb-4">

          <PacienteForm

            paciente={pacienteEditar}

            onSuccess={handleFormSuccess}

            onCancel={() => setShowForm(false)}

          />

        </div>

      )}

      {loading && <div className="alert alert-info">Cargando...</div>}

      {error && <div className="alert alert-danger">{error}</div>}

      <div className="table-responsive">

        <table className="table table-striped table-bordered align-middle">

          <thead className="table-dark">

            <tr>

              <th>Nombre</th>

              <th>Apellido</th>

              <th>NT</th>

              <th>Teléfono</th>

              <th>Email</th>

              <th>Acciones</th>

            </tr>

          </thead>

          <tbody>

            {pacientes.length === 0 && !loading ? (

              <tr>

                <td colSpan="6" className="text-center">No hay pacientes registrados</td>

              </tr>

            ) : (

              pacientes.map(paciente => (

                <tr key={paciente.\_id}>

                  <td>{paciente.nombre}</td>

                  <td>{paciente.apellido}</td>

                  <td>{paciente.dni}</td>

                  <td>{paciente.telefono}</td>

                  <td>{paciente.email}</td>

                  <td>

                    <button className="btn btn-warning btn-sm me-2" onClick={() => handleEditar(paciente)}>Editar</button>

                    <button className="btn btn-danger btn-sm" onClick={() => handleEliminar(paciente.\_id)}>Eliminar</button>

                  </td>

                </tr>

              ))

            )}

          </tbody>

        </table>

      </div>

    </div>

  );

}

export default ListaPacientes;

**////////////////////////////////////////////////////////////////////////////////////////////////**

* Propósito: Estos componentes están diseñados para manejar la entrada de datos por parte del usuario. Su función principal es mostrar un formulario (con campos de texto, selectores, fechas, etc.) que permite a los usuarios:

-Crear nuevos registros (un nuevo paciente, un nuevo médico, una nueva cita).

-Editar registros existentes.

* Cómo funcionan: Recolectan la información que el usuario ingresa en los campos del formulario y, una vez que el usuario confirma (por ejemplo, haciendo clic en "Guardar" o "Crear"), envían esos datos al backend (a través de Axios) para que sean almacenados o actualizados en la base de datos. También manejan la lógica de validación de los campos y muestran mensajes de error si es necesario.

**Cita.from.jsx**

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function CitaForm({ cita, onSuccess, onCancel }) {

  const [form, setForm] = useState({

    fecha: '',

    hora: '',

    paciente: '',

    medico: '',

    motivo: '',

    estado: 'pendiente'

  });

  const [pacientes, setPacientes] = useState([]);

  const [medicos, setMedicos] = useState([]);

  const [error, setError] = useState(null);

  const [loading, setLoading] = useState(false);

  useEffect(() => {

    axios.get('http://localhost:4000/api/pacientes').then(res => setPacientes(res.data));

    axios.get('http://localhost:4000/api/medicos').then(res => setMedicos(res.data));

    if (cita) {

      setForm({

        fecha: cita.fecha ? cita.fecha.substring(0, 10) : '',

        hora: cita.hora || '',

        paciente: cita.paciente?.\_id || cita.paciente || '',

        medico: cita.medico?.\_id || cita.medico || '',

        motivo: cita.motivo || '',

        estado: cita.estado || 'pendiente'

      });

    }

  }, [cita]);

  const handleChange = e => {

    setForm({ ...form, [e.target.name]: e.target.value });

  };

  const handleSubmit = async e => {

    e.preventDefault();

    setLoading(true);

    setError(null);

    try {

      if (cita && cita.\_id) {

        await axios.put(`http://localhost:4000/api/citas/${cita.\_id}`, form);

      } else {

        await axios.post('http://localhost:4000/api/citas', form);

      }

      setLoading(false);

      if (onSuccess) onSuccess();

    } catch (err) {

      setError('Error al guardar la cita');

      setLoading(false);

    }

  };

  return (

    <form onSubmit={handleSubmit} className="p-3 border rounded bg-light">

      <h4>{cita ? 'Editar Cita Médica' : 'Agendar Cita Médica'}</h4>

      {error && <div className="alert alert-danger">{error}</div>}

      <div className="mb-2">

        <label className="form-label">Fecha</label>

        <input type="date" className="form-control" name="fecha" value={form.fecha} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Hora</label>

        <input type="time" className="form-control" name="hora" value={form.hora} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Paciente</label>

        <select className="form-select" name="paciente" value={form.paciente} onChange={handleChange} required>

          <option value="">Seleccione un paciente</option>

          {pacientes.map(p => (

            <option key={p.\_id} value={p.\_id}>{p.nombre} {p.apellido}</option>

          ))}

        </select>

      </div>

      <div className="mb-2">

        <label className="form-label">Médico</label>

        <select className="form-select" name="medico" value={form.medico} onChange={handleChange} required>

          <option value="">Seleccione un médico</option>

          {medicos.map(m => (

            <option key={m.\_id} value={m.\_id}>{m.nombre} {m.apellido} ({m.especialidad})</option>

          ))}

        </select>

      </div>

      <div className="mb-2">

        <label className="form-label">Motivo</label>

        <input type="text" className="form-control" name="motivo" value={form.motivo} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Estado</label>

        <select className="form-select" name="estado" value={form.estado} onChange={handleChange} required>

          <option value="pendiente">Pendiente</option>

          <option value="confirmada">Confirmada</option>

          <option value="cancelada">Cancelada</option>

        </select>

      </div>

      <div className="d-flex justify-content-end gap-2 mt-3">

        <button type="button" className="btn btn-secondary" onClick={onCancel}>Cancelar</button>

        <button type="submit" className="btn btn-primary" disabled={loading}>{loading ? 'Guardando...' : 'Guardar'}</button>

      </div>

    </form>

  );

}

export default CitaForm;

**Medicofrom.jsx**

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function MedicoForm({ medico, onSuccess, onCancel }) {

  const [form, setForm] = useState({

    nombre: '',

    apellido: '',

    especialidad: '',

    telefono: '',

    email: '',

    horario: ''

  });

  const [error, setError] = useState(null);

  const [loading, setLoading] = useState(false);

  useEffect(() => {

    if (medico) {

      setForm({

        nombre: medico.nombre || '',

        apellido: medico.apellido || '',

        especialidad: medico.especialidad || '',

        telefono: medico.telefono || '',

        email: medico.email || '',

        horario: medico.horario || ''

      });

    }

  }, [medico]);

  const handleChange = e => {

    setForm({ ...form, [e.target.name]: e.target.value });

  };

  const handleSubmit = async e => {

    e.preventDefault();

    setLoading(true);

    setError(null);

    try {

      if (medico && medico.\_id) {

        await axios.put(`http://localhost:4000/api/medicos/${medico.\_id}`, form);

      } else {

        await axios.post('http://localhost:4000/api/medicos', form);

      }

      setLoading(false);

      if (onSuccess) onSuccess();

    } catch (err) {

      setError('Error al guardar el médico');

      setLoading(false);

    }

  };

  return (

    <form onSubmit={handleSubmit} className="p-3 border rounded bg-light">

      <h4>{medico ? 'Editar Médico' : 'Agregar Médico'}</h4>

      {error && <div className="alert alert-danger">{error}</div>}

      <div className="mb-2">

        <label className="form-label">Nombre</label>

        <input type="text" className="form-control" name="nombre" value={form.nombre} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Apellido</label>

        <input type="text" className="form-control" name="apellido" value={form.apellido} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Especialidad</label>

        <input type="text" className="form-control" name="especialidad" value={form.especialidad} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Teléfono</label>

        <input type="text" className="form-control" name="telefono" value={form.telefono} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Email</label>

        <input type="email" className="form-control" name="email" value={form.email} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Horario</label>

        <input type="text" className="form-control" name="horario" value={form.horario} onChange={handleChange} required />

      </div>

      <div className="d-flex justify-content-end gap-2 mt-3">

        <button type="button" className="btn btn-secondary" onClick={onCancel}>Cancelar</button>

        <button type="submit" className="btn btn-primary" disabled={loading}>{loading ? 'Guardando...' : 'Guardar'}</button>

      </div>

    </form>

  );

}

export default MedicoForm;

**Pacientefrom.jsx**

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function PacienteForm({ paciente, onSuccess, onCancel }) {

  const [form, setForm] = useState({

    nombre: '',

    apellido: '',

    dni: '',

    telefono: '',

    email: ''

  });

  const [error, setError] = useState(null);

  const [loading, setLoading] = useState(false);

  useEffect(() => {

    if (paciente) {

      setForm({

        nombre: paciente.nombre || '',

        apellido: paciente.apellido || '',

        dni: paciente.dni || '',

        telefono: paciente.telefono || '',

        email: paciente.email || ''

      });

    }

  }, [paciente]);

  const handleChange = e => {

    setForm({ ...form, [e.target.name]: e.target.value });

  };

  const handleSubmit = async e => {

    e.preventDefault();

    setLoading(true);

    setError(null);

    try {

      if (paciente && paciente.\_id) {

        await axios.put(`http://localhost:4000/api/pacientes/${paciente.\_id}`, form);

      } else {

        await axios.post('http://localhost:4000/api/pacientes', form);

      }

      setLoading(false);

      if (onSuccess) onSuccess();

    } catch (err) {

      setError('Error al guardar el paciente');

      setLoading(false);

    }

  };

  return (

    <form onSubmit={handleSubmit} className="p-3 border rounded bg-light">

      <h4>{paciente ? 'Editar Paciente' : 'Agregar Paciente'}</h4>

      {error && <div className="alert alert-danger">{error}</div>}

      <div className="mb-2">

        <label className="form-label">Nombre</label>

        <input type="text" className="form-control" name="nombre" value={form.nombre} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Apellido</label>

        <input type="text" className="form-control" name="apellido" value={form.apellido} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">NT</label>

        <input type="text" className="form-control" name="dni" value={form.dni} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Teléfono</label>

        <input type="text" className="form-control" name="telefono" value={form.telefono} onChange={handleChange} required />

      </div>

      <div className="mb-2">

        <label className="form-label">Email</label>

        <input type="email" className="form-control" name="email" value={form.email} onChange={handleChange} required />

      </div>

      <div className="d-flex justify-content-end gap-2 mt-3">

        <button type="button" className="btn btn-secondary" onClick={onCancel}>Cancelar</button>

        <button type="submit" className="btn btn-primary" disabled={loading}>{loading ? 'Guardando...' : 'Guardar'}</button>

      </div>

    </form>

  );

}

export default PacienteForm;

//////////////////////////////////////////////////////////////////////////////////////////////////////////

**>ReportePDF.jsx**

**Propósito: Este componente estaba diseñado originalmente para ser una interfaz de usuario que permitía al usuario seleccionar un tipo de reporte y un rango de fechas para generar un PDF.**

"use client"

import { useState } from "react"

import axios from "axios"

import "./ReportePDF.css"

function ReportePDF({ usuario }) {

  const [loading, setLoading] = useState(false)

  const [tipoReporte, setTipoReporte] = useState("citas")

  const [fechaInicio, setFechaInicio] = useState("")

  const [fechaFin, setFechaFin] = useState("")

  const [error, setError] = useState(null)

  const generarReporte = async () => {

    setLoading(true)

    setError(null)

    try {

      const token = localStorage.getItem("token")

      const response = await axios.post(

        "http://localhost:4000/api/reportes/generar",

        {

          tipo: tipoReporte,

          fechaInicio,

          fechaFin,

          usuarioId: usuario.entidadId || usuario.id,

          tipoUsuario: usuario.tipo,

        },

        {

          headers: { Authorization: `Bearer ${token}` },

        },

      )

      // Abrir el reporte en una nueva ventana

      const nuevaVentana = window.open("", "\_blank")

      nuevaVentana.document.write(response.data.html)

      nuevaVentana.document.close()

      // Opcional: También mostrar los datos en consola para debug

      console.log("Datos del reporte:", response.data.datos)

    } catch (err) {

      setError(err.response?.data?.error || "Error al generar el reporte")

    } finally {

      setLoading(false)

    }

  }

  return (

    <div className="reporte-container">

      <div className="reporte-header">

        <h2>

          <i className="fas fa-file-alt"></i>

          Generar Reportes

        </h2>

        <p>Genera reportes detallados que puedes imprimir o guardar como PDF</p>

      </div>

      {error && (

        <div className="error-message">

          <i className="fas fa-exclamation-circle"></i>

          <span>{error}</span>

        </div>

      )}

      <div className="reporte-form">

        <div className="form-group">

          <label htmlFor="tipoReporte">Tipo de Reporte</label>

          <select

            id="tipoReporte"

            value={tipoReporte}

            onChange={(e) => setTipoReporte(e.target.value)}

            className="form-control"

          >

            {usuario.tipo === "admin" && (

              <>

                <option value="citas">Reporte de Citas</option>

                <option value="pacientes">Reporte de Pacientes</option>

                <option value="medicos">Reporte de Médicos</option>

                <option value="estadisticas">Estadísticas Generales</option>

              </>

            )}

            {usuario.tipo === "medico" && (

              <>

                <option value="mis-citas">Mis Citas</option>

                <option value="mis-pacientes">Mis Pacientes</option>

              </>

            )}

            {usuario.tipo === "paciente" && (

              <>

                <option value="mis-citas">Mis Citas</option>

                <option value="historial">Mi Historial Médico</option>

              </>

            )}

          </select>

        </div>

        <div className="form-row">

          <div className="form-group">

            <label htmlFor="fechaInicio">Fecha Inicio (opcional)</label>

            <input

              type="date"

              id="fechaInicio"

              value={fechaInicio}

              onChange={(e) => setFechaInicio(e.target.value)}

              className="form-control"

            />

          </div>

          <div className="form-group">

            <label htmlFor="fechaFin">Fecha Fin (opcional)</label>

            <input

              type="date"

              id="fechaFin"

              value={fechaFin}

              onChange={(e) => setFechaFin(e.target.value)}

              className="form-control"

            />

          </div>

        </div>

        <button onClick={generarReporte} disabled={loading} className="generar-btn">

          {loading ? (

            <>

              <div className="spinner"></div>

              <span>Generando...</span>

            </>

          ) : (

            <>

              <i className="fas fa-file-alt"></i>

              <span>Generar Reporte</span>

            </>

          )}

        </button>

      </div>

      <div className="reporte-info">

        <h3>Información del Reporte</h3>

        <div class="info-grid">

          <div className="info-item">

            <i className="fas fa-user"></i>

            <span>

              Usuario: {usuario.nombre} {usuario.apellido}

            </span>

          </div>

          <div className="info-item">

            <i className="fas fa-calendar"></i>

            <span>Fecha: {new Date().toLocaleDateString()}</span>

          </div>

          <div className="info-item">

            <i className="fas fa-print"></i>

            <span>Formato: HTML (imprimible como PDF)</span>

          </div>

        </div>

        <div className="instrucciones">

          <h4>📋 Instrucciones:</h4>

          <ol>

            <li>Selecciona el tipo de reporte que deseas generar</li>

            <li>Opcionalmente, filtra por fechas</li>

            <li>Haz clic en "Generar Reporte"</li>

            <li>Se abrirá una nueva ventana con el reporte</li>

            <li>Usa Ctrl+P para imprimir o guardar como PDF</li>

          </ol>

        </div>

      </div>

    </div>

  )

}

export default ReportePDF

**ReportesPDF.css**

.reporte-container {

  background: white;

  border-radius: 15px;

  padding: 2rem;

  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.05);

  max-width: 600px;

  margin: 0 auto;

}

.reporte-header {

  text-align: center;

  margin-bottom: 2rem;

}

.reporte-header h2 {

  font-size: 1.5rem;

  font-weight: 700;

  color: #333;

  display: flex;

  align-items: center;

  justify-content: center;

  gap: 0.5rem;

  margin-bottom: 0.5rem;

}

.reporte-header h2 i {

  color: #667eea;

}

.reporte-header p {

  color: #666;

}

.error-message {

  background: rgba(255, 107, 107, 0.1);

  border-left: 4px solid #ff6b6b;

  color: #ff6b6b;

  padding: 1rem;

  border-radius: 8px;

  margin-bottom: 1.5rem;

  display: flex;

  align-items: center;

  gap: 0.5rem;

}

.reporte-form {

  margin-bottom: 2rem;

}

.form-group {

  margin-bottom: 1.5rem;

}

.form-group label {

  display: block;

  font-weight: 600;

  color: #333;

  margin-bottom: 0.5rem;

}

.form-control {

  width: 100%;

  padding: 0.75rem;

  border: 2px solid #e9ecef;

  border-radius: 10px;

  font-size: 1rem;

  transition: all 0.3s ease;

}

.form-control:focus {

  border-color: #667eea;

  box-shadow: 0 0 0 3px rgba(102, 126, 234, 0.25);

  outline: none;

}

.form-row {

  display: grid;

  grid-template-columns: 1fr 1fr;

  gap: 1rem;

}

.generar-btn {

  width: 100%;

  background: linear-gradient(135deg, #667eea, #764ba2);

  color: white;

  border: none;

  border-radius: 12px;

  padding: 1rem 1.5rem;

  font-weight: 600;

  cursor: pointer;

  transition: all 0.3s ease;

  display: flex;

  align-items: center;

  justify-content: center;

  gap: 0.5rem;

  box-shadow: 0 5px 15px rgba(102, 126, 234, 0.3);

}

.generar-btn:hover {

  transform: translateY(-3px);

  box-shadow: 0 8px 20px rgba(102, 126, 234, 0.4);

}

.generar-btn:disabled {

  opacity: 0.7;

  cursor: not-allowed;

  transform: none;

}

.spinner {

  width: 20px;

  height: 20px;

  border: 3px solid rgba(255, 255, 255, 0.3);

  border-radius: 50%;

  border-top-color: white;

  animation: spin 1s ease-in-out infinite;

}

@keyframes spin {

  to {

    transform: rotate(360deg);

  }

}

.reporte-info {

  border-top: 1px solid #e9ecef;

  padding-top: 1.5rem;

}

.reporte-info h3 {

  font-size: 1.2rem;

  font-weight: 700;

  color: #333;

  margin-bottom: 1rem;

}

.info-grid {

  display: flex;

  flex-direction: column;

  gap: 0.75rem;

  margin-bottom: 1.5rem;

}

.info-item {

  display: flex;

  align-items: center;

  gap: 0.75rem;

  color: #666;

  font-weight: 500;

}

.info-item i {

  color: #667eea;

  width: 20px;

}

.instrucciones {

  background: rgba(102, 126, 234, 0.1);

  border-radius: 10px;

  padding: 1.5rem;

  margin-top: 1.5rem;

}

.instrucciones h4 {

  color: #667eea;

  margin-bottom: 1rem;

  font-size: 1.1rem;

}

.instrucciones ol {

  color: #666;

  line-height: 1.6;

  margin: 0;

  padding-left: 1.5rem;

}

.instrucciones li {

  margin-bottom: 0.5rem;

}

@media (max-width: 768px) {

  .form-row {

    grid-template-columns: 1fr;

  }

  .reporte-container {

    padding: 1.5rem;

  }

}