INFORME PRUEBA PRÁCTICA

PORTADA

Tema:	Informe Prueba Práctica
Unidad de Organización Curricular:	PROFESIONAL
Nivel y Paralelo:	Quinto A
Alumnos participantes:	Bayas Axel
	Salazar Sebastián
	Daniel Moyolema
Asignatura:	Sistemas de bases de datos distribuidas
Docente:	Ing. José Caiza, Mg.

ÍNDICE

INTRODUCCIÓN

Este informe presenta el desarrollo de una aplicación distribuida orientada a la gestión de un sistema hospitalario compuesto por múltiples centros médicos. El proyecto se construyó utilizando arquitectura MVC, bases de datos distribuidas y tecnologías como contenedores Docker y Azure DevOps. La aplicación expone servicios RESTful y enfrenta retos propios de un sistema de información escalable, replicado y seguro.

OBJETIVOS

Objetivo general

Desarrollar una solución distribuida que administre información médica desde varios centros de atención, garantizando replicación, sincronización y disponibilidad de datos.

Objetivos específicos

- Configurar una infraestructura distribuida basada en VMs y Docker.
- Implementar replicación activa/pasiva y sincronización sincrónica/asincrónica.
- Diseñar una aplicación backend con Node.js y Express bajo arquitectura MVC.
- Desarrollar interfaces frontend para usuarios administrativos y médicos.
- Desplegar y documentar el sistema usando Azure DevOps.

METODOLOGÍA

- Se utilizaron tres máquinas virtuales que simulan diferentes centros médicos: Quito, Guayaquil y Cuenca.
- Se configuró MariaDB en cada nodo con esquemas de replicación.
- Se desarrolló el backend con Node.js y Express siguiendo el patrón MVC.
- La API se documentó con Swagger.
- Se diseñaron dos frontends: uno administrativo (Vue.js) y otro para hospitales (React).
- Se usaron contenedores Docker y Azure DevOps para el despliegue y automatización CI/CD.
- Se aplicaron técnicas de fragmentación horizontal, vertical e híbrida para optimizar la distribución de datos.

DESARROLLO DEL PROYECTO

Infraestructura:

- Creación de 3 VMs conectadas por red virtual (VNet) con reglas NSG configuradas.
- Instalación de MariaDB homogénea en todos los nodos.
- Configuración de Docker para facilitar nodos alternos y despliegue rápido.

Bases de datos distribuidas:

- Se analizaron modelos centralizados y distribuidos.
- Se aplicaron esquemas de replicación activa-pasiva y sincrónica-asincrónica según las necesidades de cada centro.
- Fragmentación de datos implementada con criterio híbrido (horizontal para consultas, vertical para especialidades).

Backend (MVC):

- Implementación de modelos, controladores y respuestas JSON estructuradas.
- Endpoints para CRUD de centros médicos, médicos, especialidades, empleados y consultas.

Frontend:

- Vue.js para administración (creación, edición, eliminación, reportes).
- React para hospitales (gestión de consultas con login personalizado por centro).

Despliegue:

- Azure DevOps para control de versiones y CI/CD.
- Integración de pipelines para despliegue automático y pruebas en la nube.

RESULTADOS

- La infraestructura fue correctamente configurada y funcional.
- Se logró la replicación y sincronización de bases de datos distribuidas.
- Los endpoints de la API funcionaron según lo planificado.
- Las interfaces gráficas fueron intuitivas y eficientes.
- El proyecto fue desplegado con éxito en la nube y soporta la carga distribuida entre nodos.

CONCLUSIONES

- Se logró implementar un sistema distribuido realista y funcional para el entorno hospitalario.

- Las técnicas de replicación y fragmentación permitieron un alto rendimiento y disponibilidad.
- La integración de frontend, backend y base de datos demostró la aplicabilidad del modelo MVC en un contexto distribuido.
- Azure DevOps y Docker facilitaron el desarrollo ágil y el despliegue continuo del proyecto.

RECOMENDACIONES

- Implementar un sistema de monitoreo para la replicación y rendimiento de los nodos.
- Incorporar autenticación y roles más avanzados para acceso diferenciado.
- Explorar la posibilidad de integrar servicios de inteligencia artificial para el análisis médico.

ANEXOS

- FRONTEND

  CSS

```css
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f9;
}

.container {
    width: 80%;
    margin: 0 auto;
    padding: 20px;
}

h1 {
    text-align: center;
    color: #333;
}

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

table, th, td {
    border: 1px solid #ccc;
}

th, td {
    padding: 10px;
    text-align: left;
}

```

```css
button {
    background-color: #007bff;
    color: white;
    border: none;
    padding: 5px 10px;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

input {
    margin-top: 10px;
    padding: 5px;
    width: 100%;
    box-sizing: border-box;
}
```

HTML

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Gestión de Centros Médicos</title>
    <link rel="stylesheet" href="../Css/styles.css">
</head>
<body>
    <div class="container">
        <h1>Centros Médicos</h1>
        <button id="btn-agregar">Agregar Centro Médico</button>
        <input type="text" id="search-input" placeholder="Buscar por nombre">

        <table id="centros-medicos-table">
            <thead>
                <tr>
                    <th>Nombre</th>
                    <th>Acciones</th>
                </tr>
            </thead>
            <tbody>
                <!-- Los datos se cargarán aquí -->
            </tbody>
        </table>
    </div>

    <script type="module" src="../Javascript/apiService.js"></script>
    <script type="module" src="../Javascript/centroMedicos.js"></script>
</body>
</html>
```

JAVASCRIPT

- Apiservice

```javascript
const apiService = {
    get: async function (url) {
        const response = await fetch('https://localhost:44331/api/CentroMedicos');
        if (!response.ok) throw new Error('Error en la solicitud');
        return response.json();
    },

    post: async function (url, data) {
        const response = await fetch(`/api/CentroMedicos${url}`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(data),
        });
        if (!response.ok) throw new Error('Error al crear el centro médico');
        return response.json();
    },

    put: async function (url, data) {
        const response = await fetch(`/api/CentroMedicos${url}`, {
            method: 'PUT',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(data),
        });
        if (!response.ok) throw new Error('Error al actualizar el centro médico');
        return response.json();
    },

    delete: async function (url) {
        const response = await fetch(`/api/CentroMedicos${url}`, { method: 'DELETE' });
        if (!response.ok) throw new Error('Error al eliminar el centro médico');
        return response.json();
    }
```

- CentroMedicos

```javascript
document.addEventListener('DOMContentLoaded', function() {
    loadCentrosMedicos();

    // Evento para agregar centro médico
    document.getElementById('btn-agregar').addEventListener('click', function() {
        const nombre = prompt('Ingrese el nombre del centro médico:');
        if (nombre) {
            const formData = new FormData();
            formData.append('Nombre', nombre);
            saveCentroMedico(formData);
        }
    });

    // Evento de búsqueda
    document.getElementById('search-input').addEventListener('input', function(event) {
        const query = event.target.value.toLowerCase();
        filterCentrosMedicos(query);
    });
});

// Función para cargar centros médicos desde la API
function loadCentrosMedicos() {
    fetch('https://localhost:44331/api/CentroMedicos')
        .then(response => response.json())
        .then(data => {
            displayCentrosMedicos(data.data);
        })
        .catch(error => console.error('Error al cargar centros médicos:', error));
}

// Función para mostrar los centros médicos en la tabla
function displayCentrosMedicos(centros) {
    const tableBody = document.querySelector('#centros-medicos-table tbody');
    tableBody.innerHTML = ''; // Limpiar la tabla antes de agregar datos

    centros.forEach(centro => {
```

```
            row.innerHTML = `
                <td>${centro.Nombre}</td>
                <td>
                    <button onclick="deleteCentroMedico(${centro.Id})">Eliminar</button>
                </td>
            `;
            tableBody.appendChild(row);
        });
    }
    const centroMedico = {
        nombre: "Nuevo Centro Médico",
        direccion: "Calle Falsa 123",
        ciudad: "Ciudad Ejemplo",
        telefono: "123456789",
        createdAt: new Date().toISOString()  // Si es necesario
    };
    // Función para guardar un nuevo Centro Médico

    fetch('https://localhost:44331/api/CentroMedicos', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(centroMedico)
    })
    .then(response => response.json())
    .then(data => {
        console.log('Centro Médico creado:', data);
    })
    .catch(error => {
        console.error('Error al crear el centro médico:', error);
    });
    // Función para eliminar un Centro Médico
```

```
    // Función para eliminar un Centro Médico
    function deleteCentroMedico(id) {
        if (confirm('¿Estás seguro de que quieres eliminar este centro médico?')) {
            fetch('https://localhost:44331/api/CentroMedicos/${id}', {
                method: 'DELETE'
            })
            .then(response => response.json())
            .then(data => {
                console.log(data.message);
                loadCentrosMedicos();  // Recargar la lista
            })
            .catch(error => console.error('Error al eliminar centro médico:', error));
        }
    }

    // Función para filtrar centros médicos según el nombre
    function filterCentrosMedicos(query) {
        const rows = document.querySelectorAll('#centros-medicos-table tbody tr');
        rows.forEach(row => {
            const nombre = row.cells[0].textContent.toLowerCase();
            row.style.display = nombre.includes(query) ? '' : 'none';
        });
    }
```

- Empleados

```

// Cargar empleados
function loadEmpleados() {
    const tablaEmpleados = document.querySelector('#tabla-empleados tbody');
    tablaEmpleados.innerHTML = '<tr><td colspan="7">Cargando empleados...</td></tr>';

    fetch("https://localhost:44346/api/Empleado/Listar")
        .then(response => {
            if (!response.ok) {
                throw new Error('Error en la respuesta del servidor: ' + response.status);
            }
            return response.json();
        })
        .then(data => {
            console.log('Datos recibidos de la API:', data);

            if (!Array.isArray(data)) {
                throw new Error('La API no devolvió un array de empleados');
            }

            empleados = data;
            renderEmpleadosTable(empleados);
        })
        .catch(error => {
            console.error('Error al cargar empleados:', error);
            tablaEmpleados.innerHTML = '<tr><td colspan="7">Error al cargar empleados</td></tr>';
            showNotification('Error al cargar empleados: ' + error.message, 'error');
        });
}

function loadCentros() {
    const centroSelect = document.getElementById('centro-empleado');
    centroSelect.innerHTML = '<option value="">Seleccione un centro</option>'; // Limpia los existentes

    fetch("https://localhost:44346/api/Centros/Listar")
        .then(response => {
            if (!response.ok) throw new Error("No se pudieron obtener los centros.");
            return response.json();
        })
        .then(data => {
            data.forEach(centro => {
                const option = document.createElement('option');
                option.value = centro.centroID;
                option.textContent = centro.nombre;
                centroSelect.appendChild(option);
            });
        })
        .catch(error => {
            console.error("Error al cargar centros:", error);
            showNotification("Error al cargar centros", "error");
        });
}


// Renderizar la tabla de empleados
function renderEmpleadosTable(empleadosList) {
    const tablaEmpleados = document.querySelector('#tabla-empleados tbody');
    tablaEmpleados.innerHTML = '';

    if (empleadosList.length === 0) {
        tablaEmpleados.innerHTML = `
            <tr>
                <td colspan="7" class="no-data">No hay empleados registrados</td>
            </tr>
        `;
        return;
    }
```

```
68
69      empleadosList.forEach(empleado => {
70          // Asegúrate de que centrosMedicos está cargado
71          const centro = centrosMedicos.find(c => c.CentroID === (empleado.centroID || empleado.CentroID));
72          const centroNombre = empleado.centroMedicoNombre || 'Desconocido';
73
74
75          const row = document.createElement('tr');
76          row.innerHTML = `
77              <td>${empleado.empleadoID || empleado.EmpleadoID}</td>
78              <td>${empleado.nombre || empleado.Nombre} </td>
79              <td>${empleado.apellido || empleado.Apellido || ''} </td>
80              <td>${empleado.cargo || empleado.Cargo}</td>
81              <td>${empleado.telefono || empleado.Telefono || ''}</td>
82              <td>${empleado.email || empleado.Email || ''}</td>
83              <td>${centroNombre}</td>
84              <td class="actions">
85                  <button class="btn btn-warning btn-action edit-empleado" data-id="${empleado.empleadoID || empleado.EmpleadoID}">
86                      <i class="fas fa-edit"></i>
87                  </button>
88                  <button class="btn btn-danger btn-action delete-empleado" data-id="${empleado.empleadoID || empleado.EmpleadoID}">
89                      <i class="fas fa-trash"></i>
90                  </button>
91              </td>
92          `;
93          tablaEmpleados.appendChild(row);
94
95      });
96
97      addEmpleadosActionEvents();
98  }
99  // Añadir eventos a los botones de acción de empleados
100 function addEmpleadosActionEvents() {
101     document.querySelectorAll('.edit-empleado').forEach(btn => {
```

```
102             btn.addEventListener('click', function () {
103                 const id = parseInt(this.getAttribute('data-id'));
104
105                 fetch(`https://localhost:44346/api/Empleado/Buscar/${id}`)
106                     .then(response => {
107                         if (!response.ok) {
108                             throw new Error('No se pudo obtener el empleado');
109                         }
110                         return response.json();
111                     })
112                     .then(empleado => {
113                         openEditModal(id, empleado, 'empleado'); // Modal con datos del empleado
114                     })
115                     .catch(error => {
116                         console.error('Error al obtener empleado:', error);
117                         showNotification('No se pudo cargar el empleado', 'error');
118                     });
119             });
120         });
121
122         // Botones de eliminar
123         document.querySelectorAll('.delete-empleado').forEach(btn => {
124             btn.addEventListener('click', function() {
125                 const id = parseInt(this.getAttribute('data-id'));
126                 const empleado = empleados.find(e => e.EmpleadoID === id);
127
128                 if (empleado && confirmDelete(id, `${empleado.Nombre} ${empleado.Apellido}`, 'el empleado')) {
129                     deleteEmpleado(id);
130                 }
131             });
132         });
133     }
```

```
// Buscar empleados
function searchEmpleados(term) {
    if (!term.trim()) {
        loadEmpleados();
        return;
    }

    const lowerTerm = term.toLowerCase();
    const filteredEmpleados = empleados.filter(empleado =>
        empleado.Nombre.toLowerCase().includes(lowerTerm) ||
        empleado.Apellido.toLowerCase().includes(lowerTerm) ||
        empleado.Cargo.toLowerCase().includes(lowerTerm) ||
        empleado.Email.toLowerCase().includes(lowerTerm) ||
        empleado.Telefono.includes(term)
    );

    const tablaEmpleados = document.querySelector('#tabla-empleados tbody');
    tablaEmpleados.innerHTML = '';

    if (filteredEmpleados.length === 0) {
        tablaEmpleados.innerHTML = `
            <tr>
                <td colspan="7" class="no-data">No se encontraron resultados para "${term}"</td>
            </tr>
        `;
        return;
    }

    filteredEmpleados.forEach(empleado => {
        const centro = centrosMedicos.find(c => c.CentroID === empleado.CentroID);
        const centroNombre = centro ? centro.Nombre : 'Desconocido';

        const row = document.createElement('tr');
```

```
            },
            body: JSON.stringify(empleadoData)
        })
        .then(response => {
            if (!response.ok) {
                throw new Error('Error al agregar el empleado.');
            }
            return response.json(); // o response.text() si no devuelve el objeto creado
        })
        .then(data => {
            showNotification('Empleado agregado correctamente');
            closeModal(); // si usas un modal
            loadEmpleados(); // recarga la lista actualizada
        })
        .catch(error => {
            showNotification('Error al agregar: ' + error.message, 'error');
        });
    }


// Eliminar empleado
function deleteEmpleado(id) {
    // Filtrar el array para eliminar el empleado
    empleados = empleados.filter(e => e.EmpleadoID !== id);

    // Mostrar notificación
    showNotification('Empleado eliminado correctamente');

    // Recargar tabla
    loadEmpleados();
}}
```

- Main

```
import apiService from './apiService.js';

// Elementos del DOM
const centrosTable = document.querySelector('#centros-medicos-table tbody');
const btnAgregar = document.getElementById('btn-agregar');
const searchInput = document.getElementById('search-input');

// Cargar Centros Médicos
async function cargarCentrosMedicos() {
    try {
        const response = await apiService.get('');
        const centros = response.data;
        renderizarCentrosMedicos(centros);
    } catch (error) {
        console.error('Error al cargar los centros médicos:', error);
    }
}

// Renderizar tabla
function renderizarCentrosMedicos(centros) {
    centrosTable.innerHTML = ''; // Limpiar tabla antes de renderizar

    centros.forEach(centro => {
        const row = document.createElement('tr');
        row.innerHTML = `
            <td>${centro.nombre}</td>
            <td>
                <button onclick="editarCentroMedico(${centro.id})">Editar</button>
                <button onclick="eliminarCentroMedico(${centro.id})">Eliminar</button>
            </td>
        `;
        centrosTable.appendChild(row);
    });
}

```

```
            }).catch(error => console.error('Error al agregar centro médico:', error));
        }
    });

    // Editar Centro Médico
    function editarCentroMedico(id) {
        const nuevoNombre = prompt('Ingrese el nuevo nombre del centro médico');
        if (nuevoNombre) {
            apiService.put(`/${id}`, { id, nombre: nuevoNombre }).then(() => {
                cargarCentrosMedicos(); // Recargar después de editar
            }).catch(error => console.error('Error al editar centro médico:', error));
        }
    }

    // Eliminar Centro Médico
    function eliminarCentroMedico(id) {
        if (confirm('¿Seguro que desea eliminar este centro médico?')) {
            apiService.delete(`/${id}`).then(() => {
                cargarCentrosMedicos(); // Recargar después de eliminar
            }).catch(error => console.error('Error al eliminar centro médico:', error));
        }
    }

    // Filtrar Centros Médicos por nombre
    searchInput.addEventListener('input', (e) => {
        const searchTerm = e.target.value.toLowerCase();
        const rows = centrosTable.getElementsByTagName('tr');
        Array.from(rows).forEach(row => {
            const nombre = row.getElementsByTagName('td')[0].textContent.toLowerCase();
            row.style.display = nombre.includes(searchTerm) ? '' : 'none';
        });
    });
```

- Médicos

```
import apiService from './apiService.js';
import { renderTable, showToast, closeModal } from './ui.js';

// Selectores del DOM
const tablaMedicos = document.getElementById('tabla-medicos');
const formMedico = document.getElementById('form-medico');
const modalTitle = document.getElementById('modal-title');
const btnSubmit = document.getElementById('btn-submit');

// Estado
let medicos = [];
let medicoEditando = null;

// Inicialización
document.addEventListener('DOMContentLoaded', () => {
    cargarMedicos();
    configurarFormulario();
});

// Cargar lista de médicos
async function cargarMedicos() {
    try {
        medicos = await apiService.get('Medicos');
        renderizarTabla();
    } catch (error) {
        showToast('Error al cargar médicos: ' + error.message, 'error');
        console.error(error);
    }
}

// Renderizar tabla
function renderizarTabla() {
    const columnas = [
        { key: 'id', title: 'ID' },
        { key: 'nombre', title: 'Nombre' },
        { key: 'apellido', title: 'Apellido' },
```

```
47  // Configurar formulario
48  function configurarFormulario() {
49      formMedico.addEventListener('submit', async (e) => {
50          e.preventDefault();
51
52          const formData = new FormData(formMedico);
53          const medicoData = Object.fromEntries(formData.entries());
54
55          try {
56              if (medicoEditando) {
57                  await apiService.put(`Medicos/${medicoEditando.id}`, medicoData);
58                  showToast('Médico actualizado correctamente', 'success');
59              } else {
60                  await apiService.post('Medicos', medicoData);
61                  showToast('Médico creado correctamente', 'success');
62              }
63
64              closeModal();
65              cargarMedicos();
66              formMedico.reset();
67              medicoEditando = null;
68          } catch (error) {
69              showToast('Error: ' + error.message, 'error');
70          }
71      });
72  }
```

```
94  // Eliminar médico
95  async function eliminarMedico(id) {
96      if (!confirm('¿Está seguro de eliminar este médico?')) return;
97
98      try {
99          await apiService.delete(`Medicos/${id}`);
100         showToast('Médico eliminado correctamente', 'success');
101         cargarMedicos();
102     } catch (error) {
103         showToast('Error al eliminar: ' + error.message, 'error');
104     }
105 }
106
107 // Función para abrir formulario vacío
108 export function abrirFormularioNuevoMedico() {
109     medicoEditando = null;
110     modalTitle.textContent = 'Nuevo Médico';
111     btnSubmit.textContent = 'Crear';
112     formMedico.reset();
113     openModal('medico-modal');
114 }
115
116 // Exportar funciones necesarias para otros archivos
117 export { cargarMedicos };
```

- HOSPITAL MANAGMENT SOLUTION

```
<Project Sdk="Microsoft.NET.Sdk">

  <Sdk Name="Aspire.AppHost.Sdk" Version="9.0.0" />

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <IsAspireHost>true</IsAspireHost>
    <UserSecretsId>bb3e83d5-513c-4c59-93f7-ac6b39982c00</UserSecretsId>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Aspire.Hosting.AppHost" Version="9.0.0" />
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.13" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.13">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Pomelo.EntityFrameworkCore.MySql" Version="8.0.3" />
    <PackageReference Include="Swashbuckle.AspNetCore" Version="8.1.1" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\HospitalManagementSystem\HospitalManagementSystem.csproj" />
  </ItemGroup>

</Project>
```

```
var builder = DistributedApplication.CreateBuilder(args);

builder.AddProject<Projects.HospitalManagementSystem>("hospitalmanagementsystem");

builder.Build().Run();
```

- HOSPITALMANAGEMENTSOLUTION.SERVICEDEFAULTS
    - Extensions

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Diagnostics.HealthChecks;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Diagnostics.HealthChecks;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.ServiceDiscovery;
using OpenTelemetry;
using OpenTelemetry.Metrics;
using OpenTelemetry.Trace;

namespace Microsoft.Extensions.Hosting;

// Adds common .NET Aspire services: service discovery, resilience, health checks, and OpenTelemetry.
// This project should be referenced by each service project in your solution.
// To learn more about using this project, see https://aka.ms/dotnet/aspire/service-defaults
```

```
public static class Extensions
{
    public static TBuilder AddServiceDefaults<TBuilder>(this TBuilder builder) where TBuilder : IHostApplicationBuilder
    {
        builder.ConfigureOpenTelemetry();

        builder.AddDefaultHealthChecks();

        builder.Services.AddServiceDiscovery();

        builder.Services.ConfigureHttpClientDefaults(http =>
        {
            // Turn on resilience by default
            http.AddStandardResilienceHandler();

            // Turn on service discovery by default
            http.AddServiceDiscovery();
        });

        // Uncomment the following to restrict the allowed schemes for service discovery.
        // builder.Services.Configure<ServiceDiscoveryOptions>(options =>
        // {
        //     options.AllowedSchemes = ["https"];
        // });

        return builder;
    }
```

```
    public static TBuilder ConfigureOpenTelemetry<TBuilder>(this TBuilder builder) where TBuilder : IHostApplicationBuilder
    {
        builder.Logging.AddOpenTelemetry(logging =>
        {
            logging.IncludeFormattedMessage = true;
            logging.IncludeScopes = true;
        });

        builder.Services.AddOpenTelemetry()
            .WithMetrics(metrics =>
            {
                metrics.AddAspNetCoreInstrumentation()
                    .AddHttpClientInstrumentation()
                    .AddRuntimeInstrumentation();
            })
            .WithTracing(tracing =>
            {
                tracing.AddSource(builder.Environment.ApplicationName)
                    .AddAspNetCoreInstrumentation()
                    // Uncomment the following line to enable gRPC instrumentation (requires the OpenTelemetry.Instrumentation.GrpcNetClient package)
                    //.AddGrpcClientInstrumentation()
                    .AddHttpClientInstrumentation();
            });

        builder.AddOpenTelemetryExporters();

        return builder;
    }
```

```
73      private static TBuilder AddOpenTelemetryExporters<TBuilder>(this TBuilder builder) where TBuilder : IHostApplicationBuilder
74      {
75          var useOtlpExporter = !string.IsNullOrWhiteSpace(builder.Configuration["OTEL_EXPORTER_OTLP_ENDPOINT"]);
76
77          if (useOtlpExporter)
78          {
79              builder.Services.AddOpenTelemetry().UseOtlpExporter();
80          }
81
82          // Uncomment the following lines to enable the Azure Monitor exporter (requires the Azure.Monitor.OpenTelemetry.AspNetCore package)
83          //if (!string.IsNullOrEmpty(builder.Configuration["APPLICATIONINSIGHTS_CONNECTION_STRING"]))
84          //{
85          //    builder.Services.AddOpenTelemetry()
86          //       .UseAzureMonitor();
87          //}
88
89          return builder;
90      }
91
92      public static TBuilder AddDefaultHealthChecks<TBuilder>(this TBuilder builder) where TBuilder : IHostApplicationBuilder
93      {
94          builder.Services.AddHealthChecks()
95              // Add a default liveness check to ensure app is responsive
96              .AddCheck("self", () => HealthCheckResult.Healthy(), ["live"]);
97
98          return builder;
99      }
```

```
101     public static WebApplication MapDefaultEndpoints(this WebApplication app)
102     {
103         // Adding health checks endpoints to applications in non-development environments has security implications.
104         // See https://aka.ms/dotnet/aspire/healthchecks for details before enabling these endpoints in non-development environments.
105         if (app.Environment.IsDevelopment())
106         {
107             // All health checks must pass for app to be considered ready to accept traffic after starting
108             app.MapHealthChecks("/health");
109
110             // Only health checks tagged with the "live" tag must pass for app to be considered alive
111             app.MapHealthChecks("/alive", new HealthCheckOptions
112             {
113                 Predicate = r => r.Tags.Contains("live")
114             });
115         }
116
117         return app;
118     }
119 }
```

- HOSPITALMANAGEMENTSYSTEM
    - CentroMedicosControlles

```
11  namespace HospitalManagementSystem.Controllers
12  {
13      [Route("api/[controller]")]
14      [ApiController]
15      public class CentroMedicosController : ControllerBase
16      {
17          private readonly HospitalContext _context;
18
19          public CentroMedicosController(HospitalContext context)
20          {
21              _context = context;
22          }
23
24          // GET: api/CentroMedicos
25          [HttpGet]
26          public async Task<ActionResult<IEnumerable<CentroMedicoReadDTO>>> GetCentrosMedicos()
27          {
28              var centrosMedicos = await _context.CentrosMedicos
29                  .Select(c => new CentroMedicoReadDTO
30                  {
31                      Id = c.Id,
32                      Nombre = c.Nombre,
33                      Direccion = c.Direccion,
34                      Telefono = c.Telefono,
35                      Ciudad = c.Ciudad
36                  })
37                  .ToListAsync();
38
39              return centrosMedicos;
40          }
41
42          // GET: api/CentroMedicos/5
43          [HttpGet("{id}")]
44          public async Task<ActionResult<CentroMedicoReadDTO>> GetCentroMedico(int id)
```

```
45          {
46              var centroMedico = await _context.CentrosMedicos
47                  .Where(c => c.Id == id)
48                  .Select(c => new CentroMedicoReadDTO
49                  {
50                      Id = c.Id,
51                      Nombre = c.Nombre,
52                      Direccion = c.Direccion,
53                      Telefono = c.Telefono,
54                      Ciudad = c.Ciudad
55                  })
56                  .FirstOrDefaultAsync();
57
58              if (centroMedico == null)
59              {
60                  return NotFound();
61              }
62
63              return centroMedico;
64          }
65
66          // PUT: api/CentroMedicos/5
67          [HttpPut("{id}")]
68          public async Task<IActionResult> PutCentroMedico(int id, CentroMedicoUpdateDTO centroMedicoDTO)
69          {
70              if (id != centroMedicoDTO.Id)
71              {
72                  return BadRequest();
73              }
74
75              var centroMedico = await _context.CentrosMedicos.FindAsync(id);
76              if (centroMedico == null)
```

```
78                      return NotFound();
79                  }
80
81                  // Mapear los datos del DTO a la entidad
82                  centroMedico.Nombre = centroMedicoDTO.Nombre;
83                  centroMedico.Direccion = centroMedicoDTO.Direccion;
84                  centroMedico.Telefono = centroMedicoDTO.Telefono;
85                  centroMedico.Ciudad = centroMedicoDTO.Ciudad;
86
87                  _context.Entry(centroMedico).State = EntityState.Modified;
88
89                  try
90                  {
91                      await _context.SaveChangesAsync();
92                  }
93                  catch (DbUpdateConcurrencyException)
94                  {
95                      if (!CentroMedicoExists(id))
96                      {
97                          return NotFound();
98                      }
99                      else
100                     {
101                         throw;
102                     }
103                 }
104
105                 return NoContent();
106             }
107
```

```
108         // POST: api/CentroMedicos
109         [HttpPost]
110         public async Task<ActionResult<CentroMedicoReadDTO>> PostCentroMedico(CentroMedicoCreateDTO centroMedicoDTO)
111         {
112             // Mapear el DTO a la entidad
113             var centroMedico = new CentroMedico
114             {
115                 Nombre = centroMedicoDTO.Nombre,
116                 Direccion = centroMedicoDTO.Direccion,
117                 Telefono = centroMedicoDTO.Telefono,
118                 Ciudad = centroMedicoDTO.Ciudad,
119                 CreatedAt = DateTime.Now // Se puede agregar la fecha de creación si es necesario
120             };
121
122             _context.CentrosMedicos.Add(centroMedico);
123             await _context.SaveChangesAsync();
124
125             // Mapear la entidad de vuelta al DTO para la respuesta
126             var centroMedicoReadDTO = new CentroMedicoReadDTO
127             {
128                 Id = centroMedico.Id,
129                 Nombre = centroMedico.Nombre,
130                 Direccion = centroMedico.Direccion,
131                 Telefono = centroMedico.Telefono,
132                 Ciudad = centroMedico.Ciudad
133             };
134
135             return CreatedAtAction("GetCentroMedico", new { id = centroMedico.Id }, centroMedicoReadDTO);
136         }
137
138         // DELETE: api/CentroMedicos/5
139         [HttpDelete("{id}")]
```

```
        public async Task<IActionResult> DeleteCentroMedico(int id)
        {
            var centroMedico = await _context.CentrosMedicos.FindAsync(id);
            if (centroMedico == null)
            {
                return NotFound();
            }

            _context.CentrosMedicos.Remove(centroMedico);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool CentroMedicoExists(int id)
        {
            return _context.CentrosMedicos.Any(e => e.Id == id);
        }
    }
}
```

- CitasController

```
public class CitasController : ControllerBase
{
    private readonly HospitalContext _context;

    // Constructor que inyecta el contexto de la base de datos
    public CitasController(HospitalContext context)
    {
        _context = context;
    }

    // GET: api/Citas
    // Este método obtiene todas las citas médicas de la base de datos.
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Cita>>> GetCitas()
    {
        // Se retorna la lista de todas las citas médicas de manera asíncrona
        return await _context.Citas.ToListAsync();
    }

    // GET: api/Citas/5
    // Este método obtiene una cita médica específica por su ID.
    [HttpGet("{id}")]
    public async Task<ActionResult<Cita>> GetCita(int id)
    {
        // Se busca la cita médica en la base de datos por su ID
        var cita = await _context.Citas.FindAsync(id);

        // Si no se encuentra la cita médica, se devuelve un NotFound
        if (cita == null)
        {
            return NotFound();
        }

```

```
        // PUT: api/Citas/5
        // Este método actualiza una cita médica existente en la base de datos.
        // Protege contra ataques de sobrecarga de datos, validando que el ID coincida.
        [HttpPut("{id}")]
        public async Task<IActionResult> PutCita(int id, Cita cita)
        {
            // Si el ID de la cita en la URL no coincide con el ID del objeto proporcionado, se devuelve un BadRequest
            if (id != cita.Id)
            {
                return BadRequest();
            }

            // Se marca la cita como modificada en el contexto de la base de datos
            _context.Entry(cita).State = EntityState.Modified;

            try
            {
                // Se guardan los cambios en la base de datos de manera asíncrona
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                // Si ocurre un error de concurrencia (por ejemplo, la cita no existe en la base de datos), se devuelve NotFound
                if (!CitaExists(id))
                {
                    return NotFound();
                }
                else
                {
                    // Si se produce otro tipo de error, se vuelve a lanzar la excepción
                    throw;
                }
```

```
        // POST: api/Citas
        // Este método crea una nueva cita médica en la base de datos.
        [HttpPost]
        public async Task<ActionResult<Cita>> PostCita(Cita cita)
        {
            // Se agrega la nueva cita médica al contexto de la base de datos
            _context.Citas.Add(cita);
            await _context.SaveChangesAsync();

            // Se devuelve una respuesta CreatedAtAction con el URI de la cita creada
            return CreatedAtAction("GetCita", new { id = cita.Id }, cita);
        }

        // DELETE: api/Citas/5
        // Este método elimina una cita médica de la base de datos por su ID.
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteCita(int id)
        {
            // Se busca la cita médica en la base de datos por su ID
            var cita = await _context.Citas.FindAsync(id);

            // Si no se encuentra la cita, se devuelve un NotFound
            if (cita == null)
            {
                return NotFound();
            }

            // Se elimina la cita de la base de datos
            _context.Citas.Remove(cita);
            await _context.SaveChangesAsync();
```

- ConsultasMedicasController

```
13      public class ConsultaMedicasController : ControllerBase
14      {
15          private readonly HospitalContext _context;
16
17          // Constructor que inyecta el contexto de la base de datos.
18          public ConsultaMedicasController(HospitalContext context)
19          {
20              _context = context;
21          }
22
23          // GET: api/ConsultaMedicas
24          // Este método obtiene todas las consultas médicas de la base de datos.
25          [HttpGet]
26          public async Task<ActionResult<IEnumerable<ConsultaMedica>>> GetConsultasMedicas()
27          {
28              // Se retorna la lista de consultas médicas de la base de datos de manera asíncrona.
29              return await _context.ConsultasMedicas.ToListAsync();
30          }
31
32          // GET: api/ConsultaMedicas/5
33          // Este método obtiene una consulta médica específica por su ID.
34          [HttpGet("{id}")]
35          public async Task<ActionResult<ConsultaMedica>> GetConsultaMedica(int id)
36          {
37              // Se busca la consulta médica en la base de datos utilizando el ID.
38              var consultaMedica = await _context.ConsultasMedicas.FindAsync(id);
39
40              // Si la consulta médica no se encuentra, se devuelve un NotFound.
41              if (consultaMedica == null)
42              {
43                  return NotFound();
44              }
45
46              // Se retorna la consulta médica encontrada.
47              return consultaMedica;
48          }
49
50          // PUT: api/ConsultaMedicas/5
51          // Este método actualiza una consulta médica existente por su ID.
52          // Para proteger contra ataques de sobrecarga de datos, se valida que el ID coincida.
53          [HttpPut("{id}")]
54          public async Task<IActionResult> PutConsultaMedica(int id, ConsultaMedica consultaMedica)
55          {
56              // Si el ID proporcionado no coincide con el ID de la consulta médica, se devuelve un BadRequest.
57              if (id != consultaMedica.Id)
58              {
59                  return BadRequest();
60              }
61
62              // Se marca la consulta médica como modificada en el contexto de la base de datos.
63              _context.Entry(consultaMedica).State = EntityState.Modified;
64
65              try
66              {
67                  // Se intentan guardar los cambios en la base de datos de manera asíncrona.
68                  await _context.SaveChangesAsync();
69              }
70              catch (DbUpdateConcurrencyException)
71              {
72                  // Si ocurre un error de concurrencia (por ejemplo, si la consulta médica no existe),
73                  // se devuelve un NotFound.
74                  if (!ConsultaMedicaExists(id))
75                  {
```

```
                // Se devuelve un NotFound.
                if (!ConsultaMedicaExists(id))
                {
                    return NotFound();
                }
                else
                {
                    // Si ocurre otro tipo de error, se vuelve a lanzar la excepción.
                    throw;
                }
            }

            // Se devuelve una respuesta NoContent para indicar que la operación fue exitosa.
            return NoContent();
        }

        // POST: api/ConsultaMedicas
        // Este método crea una nueva consulta médica en la base de datos.
        [HttpPost]
        public async Task<ActionResult<ConsultaMedica>> PostConsultaMedica(ConsultaMedica consultaMedica)
        {
            // Se agrega la nueva consulta médica al contexto de la base de datos.
            _context.ConsultasMedicas.Add(consultaMedica);
            await _context.SaveChangesAsync();

            // Se devuelve una respuesta CreatedAtAction con el URI del recurso creado y el objeto consulta médica.
            return CreatedAtAction("GetConsultaMedica", new { id = consultaMedica.Id }, consultaMedica);
        }

        // DELETE: api/ConsultaMedicas/5
        // Este método elimina una consulta médica de la base de datos por su ID.
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteConsultaMedica(int id)
```

```
        // DELETE: api/ConsultaMedicas/5
        // Este método elimina una consulta médica de la base de datos por su ID.
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteConsultaMedica(int id)
        {
            // Se busca la consulta médica en la base de datos utilizando el ID.
            var consultaMedica = await _context.ConsultasMedicas.FindAsync(id);

            // Si la consulta médica no se encuentra, se devuelve un NotFound.
            if (consultaMedica == null)
            {
                return NotFound();
            }

            // Se elimina la consulta médica de la base de datos.
            _context.ConsultasMedicas.Remove(consultaMedica);
            await _context.SaveChangesAsync();

            // Se devuelve una respuesta NoContent para indicar que la operación fue exitosa.
            return NoContent();
        }

        // Método privado que verifica si una consulta médica existe en la base de datos por su ID.
        private bool ConsultaMedicaExists(int id)
        {
            // Se verifica si existe una consulta médica con el ID dado.
            return _context.ConsultasMedicas.Any(e => e.Id == id);
        }
    }
}
```

- EmpleadosController

```
namespace HospitalManagementSystem.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class EmpleadosController : ControllerBase
    {
        private readonly HospitalContext _context;

        // Constructor que inyecta el contexto de la base de datos.
        public EmpleadosController(HospitalContext context)
        {
            _context = context;
        }

        // GET: api/Empleadoes
        // Este método obtiene la lista completa de empleados.
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Empleado>>> GetEmpleados()
        {
            // Se retorna la lista de empleados de la base de datos de manera asíncrona.
            return await _context.Empleados.ToListAsync();
        }

        // GET: api/Empleadoes/5
        // Este método obtiene un empleado por su ID.
        [HttpGet("{id}")]
        public async Task<ActionResult<Empleado>> GetEmpleado(int id)
        {
            // Se busca el empleado en la base de datos utilizando el ID.
            var empleado = await _context.Empleados.FindAsync(id);

            // Si el empleado no se encuentra, se devuelve un NotFound.
            if (empleado == null)
```

```
        // PUT: api/Empleadoes/5
        // Este método actualiza un empleado existente por su ID.
        // Para proteger contra ataques de sobrecarga de datos, se valida que el ID coincida.
        [HttpPut("{id}")]
        public async Task<IActionResult> PutEmpleado(int id, Empleado empleado)
        {
            // Si el ID proporcionado no coincide con el ID del empleado, se devuelve un BadRequest.
            if (id != empleado.Id)
            {
                return BadRequest();
            }

            // Se marca el empleado como modificado en el contexto de la base de datos.
            _context.Entry(empleado).State = EntityState.Modified;

            try
            {
                // Se intentan guardar los cambios en la base de datos de manera asíncrona.
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                // Si ocurre un error de concurrencia (por ejemplo, si el empleado no existe),
                // se devuelve un NotFound.
                if (!EmpleadoExists(id))
                {
                    return NotFound();
                }
                else
                {
                    // Si ocurre otro tipo de error, se vuelve a lanzar la excepción.
                    throw;
```

```
91          [HttpPost]
92          public async Task<ActionResult<Empleado>> PostEmpleado(Empleado empleado)
93          {
94              // Se agrega el nuevo empleado al contexto de la base de datos.
95              _context.Empleados.Add(empleado);
96              try
97              {
98                  // Se guardan los cambios de manera asíncrona en la base de datos.
99                  await _context.SaveChangesAsync();
100             }
101             catch (DbUpdateException)
102             {
103                 // Si el empleado ya existe (conflicto de clave primaria), se devuelve un conflicto.
104                 if (EmpleadoExists(empleado.Id))
105                 {
106                     return Conflict();
107                 }
108                 else
109                 {
110                     // Si ocurre otro tipo de error, se vuelve a lanzar la excepción.
111                     throw;
112                 }
113             }
114
115             // Se devuelve una respuesta CreatedAtAction con el URI del recurso creado y el objeto empleado.
116             return CreatedAtAction("GetEmpleado", new { id = empleado.Id }, empleado);
117         }
```

```
119         // DELETE: api/Empleadoes/5
120         // Este método elimina un empleado de la base de datos por su ID.
121         [HttpDelete("{id}")]
122         public async Task<IActionResult> DeleteEmpleado(int id)
123         {
124             // Se busca el empleado en la base de datos utilizando el ID.
125             var empleado = await _context.Empleados.FindAsync(id);
126
127             // Si el empleado no se encuentra, se devuelve un NotFound.
128             if (empleado == null)
129             {
130                 return NotFound();
131             }
132
133             // Se elimina el empleado de la base de datos.
134             _context.Empleados.Remove(empleado);
135             await _context.SaveChangesAsync();
136
137             // Se devuelve una respuesta NoContent para indicar que la operación fue exitosa.
138             return NoContent();
139         }
140
141         // Método privado que verifica si un empleado existe en la base de datos por su ID.
142         private bool EmpleadoExists(int id)
143         {
144             // Se verifica si existe un empleado con el ID dado.
145             return _context.Empleados.Any(e => e.Id == id);
146         }
147     }
148 }
```

- EspecialidadesController

```
11  namespace HospitalManagementSystem.Controllers
12  {
13      [Route("api/[controller]")]
14      [ApiController]
15      public class EspecialidadesController : ControllerBase
16      {
17          private readonly HospitalContext _context;
18
19          // Constructor que inyecta el contexto de la base de datos.
20          public EspecialidadesController(HospitalContext context)
21          {
22              _context = context;
23          }
24
25          // GET: api/Especialidades
26          // Este método obtiene la lista completa de especialidades.
27          [HttpGet]
28          public async Task<ActionResult<IEnumerable<EspecialidadReadDTO>>> GetEspecialidades()
29          {
30              // Obtener todas las especialidades y mapear a DTO
31              var especialidades = await _context.Especialidades
32                  .Select(e => new EspecialidadReadDTO
33                  {
34                      Id = e.Id,
35                      Nombre = e.Nombre,
36                      Descripcion = e.Descripcion
37                  }).ToListAsync();
38
39              return especialidades;
40          }
41
```

```
42          // GET: api/Especialidades/5
43          // Este método obtiene una especialidad por su ID.
44          [HttpGet("{id}")]
45          public async Task<ActionResult<EspecialidadReadDTO>> GetEspecialidad(int id)
46          {
47              var especialidad = await _context.Especialidades
48                  .Where(e => e.Id == id)
49                  .Select(e => new EspecialidadReadDTO
50                  {
51                      Id = e.Id,
52                      Nombre = e.Nombre,
53                      Descripcion = e.Descripcion
54                  })
55                  .FirstOrDefaultAsync();
56
57              if (especialidad == null)
58              {
59                  return NotFound();
60              }
61
62              return especialidad;
63          }
64
65          // PUT: api/Especialidades/5
66          // Este método actualiza una especialidad existente por su ID.
67          [HttpPut("{id}")]
68          public async Task<IActionResult> PutEspecialidad(int id, EspecialidadUpdateDTO especialidadDTO)
69          {
70              if (id != especialidadDTO.Id)
71              {
72                  return BadRequest();
73              }
74
```

```
                var especialidad = await _context.Especialidades.FindAsync(id);
                if (especialidad == null)
                {
                    return NotFound();
                }

                // Actualizar la especialidad con los nuevos datos
                especialidad.Nombre = especialidadDTO.Nombre;
                especialidad.Descripcion = especialidadDTO.Descripcion;

                _context.Entry(especialidad).State = EntityState.Modified;

                try
                {
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!EspecialidadExists(id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }

                return NoContent();
            }
```

```
                Descripcion = especialidad.Descripcion
            };

            return CreatedAtAction("GetEspecialidad", new { id = especialidad.Id }, especialidadReadDTO);
        }

        // DELETE: api/Especialidades/5
        // Este método elimina una especialidad de la base de datos por su ID.
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteEspecialidad(int id)
        {
            var especialidad = await _context.Especialidades.FindAsync(id);
            if (especialidad == null)
            {
                return NotFound();
            }

            _context.Especialidades.Remove(especialidad);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        // Método privado que verifica si una especialidad existe en la base de datos por su ID.
        private bool EspecialidadExists(int id)
        {
            return _context.Especialidades.Any(e => e.Id == id);
        }
    }
}
```

- HistorialMedicosController

```
13      public class HistorialMedicoesController : ControllerBase
14      {
15          private readonly HospitalContext _context;
16
17          // Constructor que inyecta el contexto de la base de datos.
18          public HistorialMedicoesController(HospitalContext context)
19          {
20              _context = context;
21          }
22
23          // GET: api/HistorialMedicoes
24          // Este método obtiene la lista completa de historial médico de todos los pacientes.
25          [HttpGet]
26          public async Task<ActionResult<IEnumerable<HistorialMedico>>> GetHistorialMedicos()
27          {
28              // Se retorna la lista de historiales médicos de la base de datos de manera asíncrona.
29              return await _context.HistorialMedicos.ToListAsync();
30          }
31
32          // GET: api/HistorialMedicoes/5
33          // Este método obtiene un historial médico por su ID.
34          [HttpGet("{id}")]
35          public async Task<ActionResult<HistorialMedico>> GetHistorialMedico(int id)
36          {
37              // Se busca el historial médico en la base de datos utilizando el ID.
38              var historialMedico = await _context.HistorialMedicos.FindAsync(id);
39
40              // Si el historial médico no se encuentra, se devuelve un NotFound.
41              if (historialMedico == null)
42              {
43                  return NotFound();
44              }
```

```
50          // PUT: api/HistorialMedicoes/5
51          // Este método actualiza un historial médico existente por su ID.
52          // Para proteger contra ataques de sobrecarga de datos, se valida el ID.
53          [HttpPut("{id}")]
54          public async Task<IActionResult> PutHistorialMedico(int id, HistorialMedico historialMedico)
55          {
56              // Si el ID proporcionado no coincide con el ID del historial médico, se devuelve un BadRequest.
57              if (id != historialMedico.Id)
58              {
59                  return BadRequest();
60              }
61
62              // Se marca el historial médico como modificado.
63              _context.Entry(historialMedico).State = EntityState.Modified;
64
65              try
66              {
67                  // Se intentan guardar los cambios en la base de datos.
68                  await _context.SaveChangesAsync();
69              }
70              catch (DbUpdateConcurrencyException)
71              {
72                  // Si ocurre un error de concurrencia (por ejemplo, si el historial médico no existe),
73                  // se devuelve un NotFound.
74                  if (!HistorialMedicoExists(id))
75                  {
76                      return NotFound();
77                  }
78                  else
79                  {
80                      // Si ocurre otro tipo de error, se vuelve a lanzar la excepción.
81                      throw;
```

```
            // Se devuelve una respuesta NoContent para indicar que la operación fue exitosa.
            return NoContent();
        }

        // POST: api/HistorialMedicoes
        // Este método crea un nuevo historial médico en la base de datos.
        [HttpPost]
        public async Task<ActionResult<HistorialMedico>> PostHistorialMedico(HistorialMedico historialMedico)
        {
            // Se agrega el nuevo historial médico al contexto.
            _context.HistorialMedicos.Add(historialMedico);

            // Se guardan los cambios en la base de datos de manera asíncrona.
            await _context.SaveChangesAsync();

            // Se devuelve un CreatedAtAction con el URI del recurso creado y el objeto historial médico.
            return CreatedAtAction("GetHistorialMedico", new { id = historialMedico.Id }, historialMedico);
        }

        // DELETE: api/HistorialMedicoes/5
        // Este método elimina un historial médico de la base de datos por su ID.
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteHistorialMedico(int id)
        {
            // Se busca el historial médico en la base de datos utilizando el ID.
            var historialMedico = await _context.HistorialMedicos.FindAsync(id);

            // Si el historial médico no se encuentra, se devuelve un NotFound.
            if (historialMedico == null)
            {
                return NotFound();
            }
```

```
            // Si el historial médico no se encuentra, se devuelve un NotFound.
            if (historialMedico == null)
            {
                return NotFound();
            }

            // Se elimina el historial médico de la base de datos.
            _context.HistorialMedicos.Remove(historialMedico);
            await _context.SaveChangesAsync();

            // Se devuelve una respuesta NoContent para indicar que la operación fue exitosa.
            return NoContent();
        }

        // Método privado que verifica si un historial médico existe en la base de datos por su ID.
        private bool HistorialMedicoExists(int id)
        {
            // Se busca si existe un historial médico con el ID dado.
            return _context.HistorialMedicos.Any(e => e.Id == id);
        }
    }
}
```

- MedicosController

```
24          public async Task<ActionResult<IEnumerable<MedicoResponseDTO>>> GetMedicos()
25          {
26              var medicos = await _context.Medicos
27                  .Include(m => m.Especialidad)
28                  .Include(m => m.CentroMedico)
29                  .Include(m => m.Persona) // Asegúrate de incluir la relación de Persona
30                  .Select(m => new MedicoResponseDTO
31                  {
32                      Id = m.Id,
33                      NumeroLicencia = m.NumeroLicencia,
34                      NombreCompleto = $"{m.Persona.Nombres} {m.Persona.Apellidos}",
35                      Especialidad = m.Especialidad.Nombre,
36                      CentroMedico = m.CentroMedico.Nombre
37                  })
38                  .ToListAsync();
39
40              return medicos;
41          }
42
43          // GET: api/Medicos/5
44          [HttpGet("{id}")]
45          public async Task<ActionResult<MedicoResponseDTO>> GetMedico(int id)
46          {
47              var medico = await _context.Medicos
48                  .Include(m => m.Especialidad)
49                  .Include(m => m.CentroMedico)
50                  .Include(m => m.Persona) // Asegúrate de incluir la relación de Persona
51                  .FirstOrDefaultAsync(m => m.Id == id);
52
53              if (medico == null)
54              {
55                  return NotFound();
```

```
58              var medicoDTO = new MedicoResponseDTO
59              {
60                  Id = medico.Id,
61                  NumeroLicencia = medico.NumeroLicencia,
62                  NombreCompleto = $"{medico.Persona.Nombres} {medico.Persona.Apellidos}",
63                  Especialidad = medico.Especialidad.Nombre,
64                  CentroMedico = medico.CentroMedico.Nombre
65              };
66
67              return medicoDTO;
68          }
69
70          // PUT: api/Medicos/5
71          [HttpPut("{id}")]
72          public async Task<IActionResult> PutMedico(int id, MedicoCreateDTO medicoDTO)
73          {
74              if (id != medicoDTO.PersonaId)
75              {
76                  return BadRequest();
77              }
78
79              var medico = await _context.Medicos.FindAsync(id);
80              if (medico == null)
81              {
82                  return NotFound();
83              }
84
85              // Actualizar los datos del médico
86              medico.NumeroLicencia = medicoDTO.NumeroLicencia;
87              medico.EspecialidadId = medicoDTO.EspecialidadId;
88              medico.CentroMedicoId = medicoDTO.CentroMedicoId;
```

```
            // Si el DTO tiene datos de Persona, actualizarlos también
            if (medicoDTO.PersonaId != null)
            {
                var persona = await _context.Personas.FindAsync(medicoDTO.PersonaId);
                if (persona != null)
                {
                    medico.Persona = persona;
                }
            }

            _context.Entry(medico).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!MedicoExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

```

```
            // DELETE: api/Medicos/5
            [HttpDelete("{id}")]
            public async Task<IActionResult> DeleteMedico(int id)
            {
                var medico = await _context.Medicos.FindAsync(id);
                if (medico == null)
                {
                    return NotFound();
                }

                _context.Medicos.Remove(medico);
                await _context.SaveChangesAsync();

                return NoContent();
            }

            private bool MedicoExists(int id)
            {
                return _context.Medicos.Any(e => e.Id == id);
            }
        }
    }
```

- PacientesController

```
namespace HospitalManagementSystem.Controllers
{
    // Definición de la ruta del controlador API
    [Route("api/[controller]")]
    [ApiController]
    public class PacientesController : ControllerBase
    {
        // Contexto de la base de datos que se inyecta mediante el constructor
        private readonly HospitalContext _context;

        // Constructor del controlador que recibe el contexto de la base de datos
        public PacientesController(HospitalContext context)
        {
            _context = context;
        }

        // GET: api/Pacientes
        // Acción para obtener todos los pacientes
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Paciente>>> GetPacientes()
        {
            // Devuelve la lista de pacientes de manera asíncrona
            return await _context.Pacientes.ToListAsync();
        }

        // GET: api/Pacientes/5
        // Acción para obtener un paciente por su ID
        [HttpGet("{id}")]
        public async Task<ActionResult<Paciente>> GetPaciente(int id)
        {
            // Busca al paciente por su ID
            var paciente = await _context.Pacientes.FindAsync(id);

            // Si el paciente no se encuentra, devuelve un error 404 (Not Found)
```

```
            if (paciente == null)
            {
                return NotFound();
            }

            // Devuelve el paciente encontrado
            return paciente;
        }

        // PUT: api/Pacientes/5
        // Acción para actualizar un paciente existente
        // Para proteger contra ataques de sobrecarga de datos (overposting), se recomienda verificar que el ID coincida con el del usuario
        [HttpPut("{id}")]
        public async Task<IActionResult> PutPaciente(int id, Paciente paciente)
        {
            // Si el ID del paciente no coincide con el ID proporcionado en la URL, devuelve un error 400 (Bad Request)
            if (id != paciente.Id)
            {
                return BadRequest();
            }

            // Marca la entrada del paciente como modificada para que se guarden los cambios en la base de datos
            _context.Entry(paciente).State = EntityState.Modified;

            try
            {
                // Intenta guardar los cambios en la base de datos
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                // Si ocurre un conflicto de concurrencia (por ejemplo, si otro usuario modificó el mismo registro), verifica si el paciente existe
```

```
            // Devuelve una respuesta sin contenido (204 No Content) si la actualización es exitosa
            return NoContent();
        }

        // POST: api/Pacientes
        // Acción para crear un nuevo paciente
        // Se utiliza el método POST para crear un nuevo recurso en la base de datos
        [HttpPost]
        public async Task<ActionResult<Paciente>> PostPaciente(Paciente paciente)
        {
            // Agrega el nuevo paciente a la base de datos
            _context.Pacientes.Add(paciente);
            try
            {
                // Intenta guardar los cambios en la base de datos
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateException)
            {
                // Si ya existe un paciente con el mismo ID, devuelve un error de conflicto (409 Conflict)
                if (PacienteExists(paciente.Id))
                {
                    return Conflict();
                }
                else
                {
                    throw;
                }
            }
```

```

            // Devuelve el paciente creado con una ruta que permite obtener ese recurso
            return CreatedAtAction("GetPaciente", new { id = paciente.Id }, paciente);
        }

        // DELETE: api/Pacientes/5
        // Acción para eliminar un paciente por su ID
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeletePaciente(int id)
        {
            // Busca al paciente por su ID
            var paciente = await _context.Pacientes.FindAsync(id);
            if (paciente == null)
            {
                // Si el paciente no existe, devuelve un error 404 (Not Found)
                return NotFound();
            }

            // Elimina al paciente de la base de datos
            _context.Pacientes.Remove(paciente);
            await _context.SaveChangesAsync();

            // Devuelve una respuesta sin contenido (204 No Content) si la eliminación es exitosa
            return NoContent();
        }

        // Método privado para verificar si un paciente existe en la base de datos
        private bool PacienteExists(int id)
        {
            // Devuelve verdadero si el paciente con el ID especificado existe en la base de datos
            return _context.Pacientes.Any(e => e.Id == id);
        }
    }
```

- PersonasController

```
namespace HospitalManagementSystem.Controllers
{
    // Definición de la ruta del controlador API
    [Route("api/[controller]")]
    [ApiController]
    public class PersonasController : ControllerBase
    {
        // Contexto de la base de datos que se inyecta mediante el constructor
        private readonly HospitalContext _context;

        // Constructor del controlador que recibe el contexto de la base de datos
        public PersonasController(HospitalContext context)
        {
            _context = context;
        }

        // GET: api/Personas
        // Acción para obtener todas las personas
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Persona>>> GetPersonas()
        {
            // Devuelve la lista de personas de manera asíncrona
            return await _context.Personas.ToListAsync();
        }

        // GET: api/Personas/5
        // Acción para obtener una persona por su ID
        [HttpGet("{id}")]
        public async Task<ActionResult<Persona>> GetPersona(int id)
        {
            // Busca a la persona por su ID
            var persona = await _context.Personas.FindAsync(id);
```

```

            // Si la persona no se encuentra, devuelve un error 404 (Not Found)
            if (persona == null)
            {
                return NotFound();
            }

            // Devuelve la persona encontrada
            return persona;
        }

        // PUT: api/Personas/5
        // Acción para actualizar una persona existente
        // Para proteger contra ataques de sobrecarga de datos (overposting), se recomienda verificar que el ID coincida con el del usuario
        [HttpPut("{id}")]
        public async Task<IActionResult> PutPersona(int id, Persona persona)
        {
            // Si el ID de la persona no coincide con el ID proporcionado en la URL, devuelve un error 400 (Bad Request)
            if (id != persona.Id)
            {
                return BadRequest();
            }

            // Marca la entrada de la persona como modificada para que se guarden los cambios en la base de datos
            _context.Entry(persona).State = EntityState.Modified;

            try
            {
                // Intenta guardar los cambios en la base de datos
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
```

```
                {
                    // Si ocurre un conflicto de concurrencia (por ejemplo, si otro usuario modificó el mismo registro), verifica si la persona
                    if (!PersonaExists(id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }

                // Devuelve una respuesta sin contenido (204 No Content) si la actualización es exitosa
                return NoContent();
            }

            // POST: api/Personas
            // Acción para crear una nueva persona
            // Se utiliza el método POST para crear un nuevo recurso en la base de datos
            [HttpPost]
            public async Task<ActionResult<Persona>> PostPersona(Persona persona)
            {
                // Agrega la nueva persona a la base de datos
                _context.Personas.Add(persona);
                await _context.SaveChangesAsync();

                // Devuelve la persona creada con una ruta que permite obtener ese recurso
                return CreatedAtAction("GetPersona", new { id = persona.Id }, persona);
            }

            // DELETE: api/Personas/5
            // Acción para eliminar una persona por su ID
            [HttpDelete("{id}")]
```

```
            public async Task<IActionResult> DeletePersona(int id)
            {
                // Busca a la persona por su ID
                var persona = await _context.Personas.FindAsync(id);
                if (persona == null)
                {
                    // Si la persona no existe, devuelve un error 404 (Not Found)
                    return NotFound();
                }

                // Elimina la persona de la base de datos
                _context.Personas.Remove(persona);
                await _context.SaveChangesAsync();

                // Devuelve una respuesta sin contenido (204 No Content) si la eliminación es exitosa
                return NoContent();
            }

            // Método privado para verificar si una persona existe en la base de datos
            private bool PersonaExists(int id)
            {
                // Devuelve verdadero si la persona con el ID especificado existe en la base de datos
                return _context.Personas.Any(e => e.Id == id);
            }
        }
    }
```

- UsuariosController

```
8
9   namespace HospitalManagementSystem.Controllers
10  {
11      // Definición de la ruta del controlador API
12      [Route("api/[controller]")]
13      [ApiController]
14      public class UsuariosController : ControllerBase
15      {
16          // Contexto de la base de datos que se inyecta mediante el constructor
17          private readonly HospitalContext _context;
18
19          // Constructor del controlador que recibe el contexto de la base de datos
20          public UsuariosController(HospitalContext context)
21          {
22              _context = context;
23          }
24
25          // GET: api/Usuarios
26          // Acción para obtener todos los usuarios
27          [HttpGet]
28          public async Task<ActionResult<IEnumerable<Usuario>>> GetUsuarios()
29          {
30              // Devuelve la lista de usuarios de manera asíncrona
31              return await _context.Usuarios.ToListAsync();
32          }
33
34          // GET: api/Usuarios/5
35          // Acción para obtener un usuario por su ID
36          [HttpGet("{id}")]
37          public async Task<ActionResult<Usuario>> GetUsuario(int id)
38          {
39              // Busca al usuario por su ID
40              var usuario = await _context.Usuarios.FindAsync(id);
41
42              // Si el usuario no se encuentra, devuelve un error 404 (Not Found)
```

```
43              if (usuario == null)
44              {
45                  return NotFound();
46              }
47
48              // Devuelve el usuario encontrado
49              return usuario;
50          }
51
52          // PUT: api/Usuarios/5
53          // Acción para actualizar un usuario existente
54          // Para proteger contra ataques de sobrecarga de datos (overposting), se recomienda verificar que el ID coincida con el del usuario
55          [HttpPut("{id}")]
56          public async Task<IActionResult> PutUsuario(int id, Usuario usuario)
57          {
58              // Si el ID del usuario no coincide con el ID proporcionado en la URL, devuelve un error 400 (Bad Request)
59              if (id != usuario.Id)
60              {
61                  return BadRequest();
62              }
63
64              // Marca la entrada del usuario como modificada para que se guarden los cambios en la base de datos
65              _context.Entry(usuario).State = EntityState.Modified;
66
67              try
68              {
69                  // Intenta guardar los cambios en la base de datos
70                  await _context.SaveChangesAsync();
71              }
72              catch (DbUpdateConcurrencyException)
73              {
74                  // Si ocurre un conflicto de concurrencia (por ejemplo, si otro usuario modificó el mismo registro), verifica si el usuario existe
75                  if (!UsuarioExists(id))
```

```
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            // Devuelve una respuesta sin contenido (204 No Content) si la actualización es exitosa
            return NoContent();
        }

        // POST: api/Usuarios
        // Acción para crear un nuevo usuario
        // Se utiliza el método POST para crear un nuevo recurso en la base de datos
        [HttpPost]
        public async Task<ActionResult<Usuario>> PostUsuario(Usuario usuario)
        {
            // Agrega el nuevo usuario a la base de datos
            _context.Usuarios.Add(usuario);
            await _context.SaveChangesAsync();

            // Devuelve el usuario creado con una ruta que permite obtener ese recurso
            return CreatedAtAction("GetUsuario", new { id = usuario.Id }, usuario);
        }

        // DELETE: api/Usuarios/5
        // Acción para eliminar un usuario por su ID
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteUsuario(int id)
        {
```

```
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteUsuario(int id)
        {
            // Busca al usuario por su ID
            var usuario = await _context.Usuarios.FindAsync(id);
            if (usuario == null)
            {
                // Si el usuario no existe, devuelve un error 404 (Not Found)
                return NotFound();
            }

            // Elimina el usuario de la base de datos
            _context.Usuarios.Remove(usuario);
            await _context.SaveChangesAsync();

            // Devuelve una respuesta sin contenido (204 No Content) si la eliminación es exitosa
            return NoContent();
        }

        // Método privado para verificar si un usuario existe en la base de datos
        private bool UsuarioExists(int id)
        {
            // Devuelve verdadero si el usuario con el ID especificado existe en la base de datos
            return _context.Usuarios.Any(e => e.Id == id);
        }
    }
}
```

- DATA/CONTEXT
    - HospitalContext

```
        // DbSets para cada tabla
        public DbSet<CentroMedico> CentrosMedicos { get; set; }
        public DbSet<Especialidad> Especialidades { get; set; }
        public DbSet<Persona> Personas { get; set; }
        public DbSet<Medico> Medicos { get; set; }
        public DbSet<Empleado> Empleados { get; set; }
        public DbSet<Paciente> Pacientes { get; set; }
        public DbSet<Usuario> Usuarios { get; set; }
        public DbSet<ConsultaMedica> ConsultasMedicas { get; set; }
        public DbSet<Cita> Citas { get; set; }
        public DbSet<HistorialMedico> HistorialMedicos { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            // Configuración de nombres de tablas
            modelBuilder.Entity<CentroMedico>().ToTable("centros_medicos");
            modelBuilder.Entity<Especialidad>().ToTable("especialidades");
            modelBuilder.Entity<Persona>().ToTable("personas");
            modelBuilder.Entity<Medico>().ToTable("medicos");
            modelBuilder.Entity<Empleado>().ToTable("empleados");
            modelBuilder.Entity<Paciente>().ToTable("pacientes");
            modelBuilder.Entity<Usuario>().ToTable("usuarios");
            modelBuilder.Entity<ConsultaMedica>().ToTable("consultas_medicas");
            modelBuilder.Entity<Cita>().ToTable("citas");
            modelBuilder.Entity<HistorialMedico>().ToTable("historial_medico");

            // Configuración de claves primarias
            modelBuilder.Entity<Medico>()
                .HasKey(m => m.Id);

            modelBuilder.Entity<Empleado>()
                .HasKey(e => e.Id);

```

```
            modelBuilder.Entity<Medico>()
                .HasOne(m => m.Persona)
                .WithOne(p => p.Medico)
                .HasForeignKey<Medico>(m => m.Id)
                .OnDelete(DeleteBehavior.Restrict);

            modelBuilder.Entity<Empleado>()
                .HasOne(e => e.Persona)
                .WithOne(p => p.Empleado)
                .HasForeignKey<Empleado>(e => e.Id)
                .OnDelete(DeleteBehavior.Restrict);

            modelBuilder.Entity<Paciente>()
                .HasOne(p => p.Persona)
                .WithOne(p => p.Paciente)
                .HasForeignKey<Paciente>(p => p.Id)
                .OnDelete(DeleteBehavior.Restrict);

            // Relaciones N:1
            modelBuilder.Entity<Medico>()
                .HasOne(m => m.Especialidad)
                .WithMany(e => e.Medicos)
                .HasForeignKey(m => m.EspecialidadId)
                .OnDelete(DeleteBehavior.Restrict);

            modelBuilder.Entity<Medico>()
                .HasOne(m => m.CentroMedico)
                .WithMany(c => c.Medicos)
                .HasForeignKey(m => m.CentroMedicoId)
                .OnDelete(DeleteBehavior.Restrict);

            modelBuilder.Entity<Empleado>()
                .HasOne(e => e.CentroMedico)
```

```
                .HasForeignKey(c => c.PacienteId)
                .OnDelete(DeleteBehavior.Restrict);

            modelBuilder.Entity<Cita>()
                .HasOne(c => c.CentroMedico)
                .WithMany(c => c.Citas)
                .HasForeignKey(c => c.CentroMedicoId)
                .OnDelete(DeleteBehavior.Restrict);

            // Configuración de Historial Médico
            modelBuilder.Entity<HistorialMedico>()
                .HasOne(h => h.Paciente)
                .WithMany(p => p.HistorialMedicos)
                .HasForeignKey(h => h.PacienteId)
                .OnDelete(DeleteBehavior.Restrict);

            // Configuración de Usuarios
            modelBuilder.Entity<Usuario>()
                .HasOne(u => u.Persona)
                .WithMany(p => p.Usuarios)
                .HasForeignKey(u => u.PersonaId)
                .OnDelete(DeleteBehavior.Restrict);

            // Valores por defecto
            modelBuilder.Entity<Persona>()
                .Property(p => p.CreatedAt)
                .HasDefaultValueSql("CURRENT_TIMESTAMP");

            modelBuilder.Entity<CentroMedico>()
                .Property(c => c.CreatedAt)
                .HasDefaultValueSql("CURRENT_TIMESTAMP");

            modelBuilder.Entity<Cita>()
```

```
    public override int SaveChanges()
    {
        // Actualizar automáticamente campos de auditoría
        var entries = ChangeTracker.Entries()
            .Where(e => e.Entity is BaseEntity && (
                    e.State == EntityState.Added
                    || e.State == EntityState.Modified));

        foreach (var entityEntry in entries)
        {
            ((BaseEntity)entityEntry.Entity).UpdatedAt = DateTime.Now;

            if (entityEntry.State == EntityState.Added)
            {
                ((BaseEntity)entityEntry.Entity).CreatedAt = DateTime.Now;
            }
        }

        return base.SaveChanges();
    }
}

// Clase base opcional para campos de auditoría
public abstract class BaseEntity
{
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }
}
```

- DTOS
    - CentroMedicoDto

```
5       // DTO para crear un centro médico
6       public class CentroMedicoCreateDTO
7       {
8           [Required]
9           public string Nombre { get; set; }
10
11          public string Direccion { get; set; } = string.Empty;
12
13          [Required]
14          public string Telefono { get; set; }
15
16          [Required]
17          [EmailAddress]
18          public string Ciudad { get; set; }
19      }
20
21      // DTO para mostrar info (lectura)
22      public class CentroMedicoReadDTO
23      {
24          public int Id { get; set; } // Corregido a "Id" para que coincida con la propiedad de la entidad
25
26          public string Nombre { get; set; }
27
28          public string Direccion { get; set; }
29
30          public string Telefono { get; set; }
31
32          public string Ciudad { get; set; }
33      }
34
```

```
35      // DTO para actualizar
36      public class CentroMedicoUpdateDTO
37      {
38          public int Id { get; set; }
39          [Required]
40          public string Nombre { get; set; }
41
42          public string Direccion { get; set; } = string.Empty;
43
44          [Required]
45          public string Telefono { get; set; }
46
47          [Required]
48          [EmailAddress]
49          public string Ciudad { get; set; }
50      }
51  }
```

- EspecialidadDto

```
using System.ComponentModel.DataAnnotations;

namespace HospitalManagementSystem.DTOs
{
    // DTO para crear una especialidad
    public class EspecialidadCreateDTO
    {
        [Required]
        [StringLength(100)]
        public string Nombre { get; set; }

        public string Descripcion { get; set; }
    }

    // DTO para mostrar la información de una especialidad (lectura)
    public class EspecialidadReadDTO
    {
        public int Id { get; set; }
        public string Nombre { get; set; }
        public string Descripcion { get; set; }
    }

    // DTO para actualizar una especialidad
    public class EspecialidadUpdateDTO
    {
        public int Id { get; set; }

        [Required]
        [StringLength(100)]
        public string Nombre { get; set; }
```

```
    // DTO para eliminar una especialidad (solo requiere el Id)
    public class EspecialidadDeleteDTO
    {
        public int Id { get; set; }
    }
}
```

- MedicoDto

```
namespace HospitalManagementSystem.DTOs
{
    // DTO para el registro de un nuevo Médico
    public class MedicoCreateDTO
    {
        public string NumeroLicencia { get; set; }
        public int EspecialidadId { get; set; }
        public int CentroMedicoId { get; set; }

        // Información de la persona asociada
        public int PersonaId { get; set; }
        public string Nombre { get; set; }
        public string Apellido { get; set; }
        public DateTime FechaNacimiento { get; set; }
        public string Direccion { get; set; }
        public string Telefono { get; set; }
        public string Correo { get; set; }
    }

    // DTO para la actualización de un Médico
    public class MedicoUpdateDTO
    {
        public string NumeroLicencia { get; set; }
        public int EspecialidadId { get; set; }
        public int CentroMedicoId { get; set; }

        // Información de la persona asociada
        public int PersonaId { get; set; }
        public string Nombre { get; set; }
        public string Apellido { get; set; }
```

```
// DTO para la respuesta de un Médico (con la información de la Persona asociada)
public class MedicoDTO
{
    public int Id { get; set; }
    public string NumeroLicencia { get; set; }
    public int EspecialidadId { get; set; }
    public int CentroMedicoId { get; set; }

    // Información de la persona asociada
    public int PersonaId { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public DateTime FechaNacimiento { get; set; }
    public string Direccion { get; set; }
    public string Telefono { get; set; }
    public string Correo { get; set; }
}

// DTO para eliminar un Médico y su Persona asociada
public class MedicoDeleteDTO
{
    public int MedicoId { get; set; } // Identificador del médico a eliminar
    public bool EliminarPersona { get; set; } // Indicador para saber si también eliminar la persona asociada
}

// DTO para la lista de Médicos
public class MedicoListDTO
{
    public int Id { get; set; }
    public string NumeroLicencia { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
}
```

```
62          // DTO para la lista de Médicos
63          public class MedicoListDTO
64          {
65              public int Id { get; set; }
66              public string NumeroLicencia { get; set; }
67              public string Nombre { get; set; }
68              public string Apellido { get; set; }
69          }
70
71          // DTO para la respuesta de un Médico (solo para los detalles)
72          public class MedicoResponseDTO
73          {
74              public int Id { get; set; }
75              public string NumeroLicencia { get; set; }
76              public string NombreCompleto { get; set; }
77              public string Especialidad { get; set; }
78              public string CentroMedico { get; set; }
79          }
80      }
```

- PersonaDto

```
namespace HospitalManagementSystem.DTOs
{
    public class PersonaCreateDTO
    {
        public string Nombres { get; set; }
        public string Apellidos { get; set; }
        public DateTime? FechaNacimiento { get; set; }
        public string Genero { get; set; }
        public string Telefono { get; set; }
        public string Email { get; set; }
        public string Direccion { get; set; }
        public string Cedula { get; set; }
    }

    public class PersonaUpdateDTO
    {
        public int Id { get; set; }
        public string Nombres { get; set; }
        public string Apellidos { get; set; }
        public DateTime? FechaNacimiento { get; set; }
        public string Genero { get; set; }
        public string Telefono { get; set; }
        public string Email { get; set; }
        public string Direccion { get; set; }
        public string Cedula { get; set; }
    }
```

```
    public class PersonaResponseDTO
    {
        public int Id { get; set; }
        public string Nombres { get; set; }
        public string Apellidos { get; set; }
        public DateTime? FechaNacimiento { get; set; }
        public string Genero { get; set; }
        public string Telefono { get; set; }
        public string Email { get; set; }
        public string Direccion { get; set; }
        public string Cedula { get; set; }
    }
}
```

- WWROOT

```
using Microsoft.EntityFrameworkCore;
using Pomelo.EntityFrameworkCore.MySql.Infrastructure;
using Microsoft.OpenApi.Models;
using System.Reflection;
using HospitalManagementSystem;

var builder = WebApplication.CreateBuilder(args);

// Configuración básica
builder.Configuration.AddJsonFile("appsettings.json", optional: false, reloadOnChange: true);

// Configuración CORS para puerto 5501 (Live Server/VSCode)
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowLocalhost", policy =>
    {
        policy.WithOrigins("http://127.0.0.1:5501")
              .AllowAnyHeader()
              .AllowAnyMethod();
    });
});

// Configuración de controladores
builder.Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.PropertyNamingPolicy = null;
        options.JsonSerializerOptions.WriteIndented = true;
    });

```

```
30
31  // Configuración de la base de datos
32  var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
33  builder.Services.AddDbContext<HospitalContext>(options =>
34  {
35      options.UseMySql(
36          connectionString,
37          ServerVersion.AutoDetect(connectionString),
38          mysqlOptions =>
39          {
40              mysqlOptions.EnableRetryOnFailure(
41                  maxRetryCount: 5,
42                  maxRetryDelay: TimeSpan.FromSeconds(30),
43                  errorNumbersToAdd: null);
44          });
45  });
46
47  // Configuración de Swagger
48  builder.Services.AddEndpointsApiExplorer();
49  builder.Services.AddSwaggerGen(c =>
50  {
51      c.SwaggerDoc("v1", new OpenApiInfo
52      {
53          Title = "Hospital Management API",
54          Version = "v1",
55          Description = "API para gestión hospitalaria - Desarrollo (Puerto 5501)",
56          Contact = new OpenApiContact
57          {
58              Name = "Equipo de Desarrollo",
59              Email = "desarrollo@hospital.com"
60          }
61      });
62
```

```
        var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
        var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
        if (File.Exists(xmlPath))
        {
            c.IncludeXmlComments(xmlPath);
        }

        c.CustomSchemaIds(x => x.FullName);
    });

    var app = builder.Build();

    // Configuración del entorno
    if (app.Environment.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();

        // Migraciones automáticas solo en desarrollo
        using var scope = app.Services.CreateScope();
        var dbContext = scope.ServiceProvider.GetRequiredService<HospitalContext>();
        dbContext.Database.Migrate();
    }

    // Middleware CORS (debe estar antes de otros middlewares)
    app.UseCors("AllowLocalhost");

    // Configuración Swagger UI
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Hospital API v1");
        c.RoutePrefix = "swagger";
        c.ConfigObject.AdditionalItems["syntaxHighlight"] = new Dictionary<string, object>
        {
```

```
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Hospital API v1");
        c.RoutePrefix = "swagger";
        c.ConfigObject.AdditionalItems["syntaxHighlight"] = new Dictionary<string, object>
        {
            ["activated"] = false
        };
    });

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseRouting();
    app.UseAuthorization();

    // Mapeo de endpoints
    app.MapControllers();

    // Configuración opcional para SPA fallback
    //app.MapFallbackToFile("index.html");

    app.Run();
```

DOCKERFILE

```
# Etapa de construcción
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["HospitalManagementSystem/HospitalManagementSystem.csproj", "HospitalManagementSystem/"]
RUN dotnet restore "HospitalManagementSystem/HospitalManagementSystem.csproj"
COPY . .
WORKDIR "/src/HospitalManagementSystem"
RUN dotnet build "HospitalManagementSystem.csproj" -c Release -o /app/build

# Etapa de publicación
FROM build AS publish
RUN dotnet publish "HospitalManagementSystem.csproj" -c Release -o /app/publish

# Etapa final
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS final
WORKDIR /app
COPY --from=publish /app/publish .

# Configuración de certificados (opcional, solo si necesitas HTTPS)
RUN apt-get update && \
    apt-get install -y openssl && \
    mkdir -p /app/https && \
    openssl req -x509 -newkey rsa:4096 -nodes -keyout /app/https/key.pem -out /app/https/cert.pem -subj "/CN=localhost" -days 365

ENV ASPNETCORE_URLS=https://+:443;http://+:80
ENV ASPNETCORE_Kestrel__Certificates__Default__Path=/app/https/cert.pem
ENV ASPNETCORE_Kestrel__Certificates__Default__KeyPath=/app/https/key.pem

ENTRYPOINT ["dotnet", "HospitalManagementSystem.dll"]
```