Documentación Técnica

Sistema de Gestión de Empleados (RRHH)

---

1. Introducción

El **Sistema de Gestión de Empleados (RRHH)** es una solución web desarrollada como respuesta a una prueba técnica, cuyo objetivo es **centralizar y optimizar la administración de la información de los empleados**, reemplazando procesos manuales por una aplicación **robusta, segura y mantenible**.

La solución se basa en una **arquitectura cliente-servidor**:

- **Backend**: expone una **API REST** desarrollada en **ASP.NET Core (.NET 9)**, responsable de la lógica de negocio, la persistencia de datos y la integridad de la información.
- **Frontend**: desarrollado con **Angular e Ionic**, ofrece una interfaz moderna, responsive y adaptable a distintos dispositivos, consumiendo de forma segura los servicios de la API.

---

2. Arquitectura del Sistema

El backend implementa **Clean Architecture**, garantizando que cada capa tenga una responsabilidad única y que las dependencias fluyan hacia el núcleo del dominio. Este enfoque asegura un sistema **escalable, mantenible y altamente testeable**.

2.1. RRHH.Domain (Núcleo)

Es la capa central del sistema y **no tiene dependencias externas**. Contiene las reglas de negocio fundamentales.

Responsabilidades: - **Entities**: Modelo de dominio, destacando la entidad `Employee` con los atributos: - `Id` - `FullName` - `HireDate` - `Position` - `Salary` - `Department` - **DTOs (Data Transfer Objects)**: Objetos para transportar datos entre capas sin exponer la estructura interna de la base de datos.

2.2. RRHH.Application (Contratos)

Define **qué puede hacer el sistema**, sin detallar cómo se implementa. Desacopla la lógica de negocio de la infraestructura.

Responsabilidades: - **Interfaces**: Contratos como `IEmployeeService` y `IGenericRepository`, que permiten la inversión de dependencias y facilitan las pruebas unitarias.

2.3. RRHH.Services (Lógica de Negocio)

Implementa la lógica definida en la capa de Application.

Responsabilidades: - **Servicios de Aplicación**: Clases como `EmployeeService`, encargadas de: - Validar reglas de negocio (ej. salarios mayores a cero). - Orquestar el flujo entre controladores y repositorios. - Garantizar la integridad de los datos antes de su persistencia.

2.4. RRHH.Infrastructure (Acceso a Datos)

Gestiona la comunicación con recursos externos y la persistencia de datos.

Responsabilidades: - **Data Context**: `RRHHDbContext` configurado con **Entity Framework Core**. - **Repositories**: Implementación concreta del patrón Repositorio (ej. `GenericRepository`) para operaciones CRUD sobre **SQL Server**.

2.5. RRHH.API (Presentación)

Es el punto de entrada para clientes externos como el frontend Angular/Ionic y herramientas de prueba.

Responsabilidades: - **Controllers**: Manejo de peticiones HTTP (`GET`, `POST`, `PUT`, `DELETE`). - **Configuración**: En `Program.cs` se gestiona la inyección de dependencias, middleware y configuración general de la aplicación.

---

3. Tecnologías Utilizadas

Tecnología	Uso
.NET 9	Backend
C#	Lenguaje de programación
ASP.NET Core Web API	Exposición de servicios REST
Entity Framework Core	ORM y acceso a datos
SQL Server	Base de datos relacional
Angular	Frontend web
Ionic	Interfaz de usuario responsive
Postman	Pruebas de API
Swagger / OpenAPI	Documentación y pruebas de endpoints

Tecnología	Uso
Visual Studio 2022	Entorno de desarrollo

---

4. Base de Datos

La base de datos se compone de la tabla principal **Employees**, encargada de la gestión centralizada del personal.

Diccionario de Datos – Tabla `Employees`

Campo	Tipo de dato	Descripción	Restricciones
Id	INT	Identificador único	PK, Identity (1,1)
FullName	VARCHAR(150)	Nombre completo	NOT NULL
HireDate	DATE	Fecha de contratación	NOT NULL
Position	VARCHAR(100)	Cargo	NOT NULL
Salary	DECIMAL(18,2)	Salario	NOT NULL, CHECK (> 0)
Department	VARCHAR(100)	Departamento	NOT NULL

Integridad de Datos: - Restricción `CHECK (Salary > 0)` para validar valores financieros. - Restricciones `NOT NULL` para asegurar consistencia.

---

5. Funcionalidades Implementadas (API Endpoints)

Método	Endpoint	Descripción
GET	`/api/employees?page=1&pageSize=10`	Listar empleados (paginado)
GET	`/api/employees/{id}`	Obtener empleado por ID
GET	`/api/employees/search?term={valor}`	Buscar empleado por nombre o ID
POST	`/api/employees`	Crear empleado
PUT	`/api/employees/{id}`	Actualizar empleado
DELETE	`/api/employees/{id}`	Eliminar empleado

---

6. Ejecución del Proyecto

1. **Base de Datos**: Crear la base de datos en SQL Server y ejecutar el script de la tabla `Employees`.
2. **Configuración**: Definir la cadena de conexión en `appsettings.json`.
3. **Backend**: Ejecutar el proyecto `RRHH.API`.
4. **Pruebas**: Validar los endpoints usando **Postman** o **Swagger UI**.

---

7. Estrategia de Calidad y Pruebas

7.1. Pruebas de Integración y API

- Validación de endpoints con **Postman**.
- Verificación de códigos HTTP correctos: `200`, `201`, `204`, `404`.
- Comprobación de la integridad de los datos.

7.2. Pruebas Unitarias – Frontend (Angular)

Se implementaron pruebas automatizadas con **Jasmine y Karma**.

Cobertura de Código:

Métrica	Cobertura	Estado
Lines	81.53% (159/195)	✅Meta superada
Functions	82.25%	✅Óptimo
Statements	78.64%	⚠️Aceptable
Branches	52.45%	ℹ️En proceso

Glosario: - **Lines**: Porcentaje de líneas ejecutadas. - **Functions**: Métodos invocados. - **Branches**: Caminos lógicos (`if/else`). El *camino feliz* está cubierto al 100%.

7.3. Pruebas Unitarias – Backend (.NET)

Pruebas realizadas con **Mocks** para aislar dependencias externas.

Componente	Cobertura de Líneas	Cobertura de Ramas	Estado
RRHH.Services	100% (94/94)	100% (16/16)	🔰Óptimo
RRHH.API.Controllers	100% (44/44)	100% (8/8)	🔰Óptimo

Componente	Cobertura de Líneas	Cobertura de Ramas	Estado
RRHH.Infrastructure	0%	0%	Excluido (Mock)

---

8. Oportunidades de Mejora y Roadmap

8.1. Normalización y Modelo de Datos

- Separar entidades en tablas independientes para **Departments** y **Positions**.
- Reducir redundancia y facilitar la gestión de catálogos.

8.2. Evolución Arquitectónica

- **CQRS + MediatR**: Separar operaciones de lectura y escritura.
- **Microservicios**: Preparación para escalar módulos de forma independiente si el dominio crece.

Conclusión

El Sistema de Gestión de Empleados ha sido desarrollado priorizando **buenas prácticas, calidad de código y testabilidad**. La arquitectura implementada proporciona una base sólida que resuelve la necesidad actual y permite una **evolución futura hacia soluciones empresariales de mayor escala**.

Esta solución cumple con los requerimientos técnicos planteados y demuestra un enfoque profesional en diseño, implementación y aseguramiento de calidad.

Autor: Iván Martínez Aguas