Documentación Técnica: Sistema de Gestión de Ofertas

Autor: Jhon Sebastian Bulla Rojas

Introducción

El Sistema de Gestión de Ofertases una aplicación web desarrollada utilizando ASP.NET MVC y ASP.NET Web API. Su propósito principal es gestionar ofertas, permitiendo el registro, edición, eliminación y visualización de la información de las mismas. Esta documentación técnica proporciona una visión detallada de los aspectos técnicos y de diseño del sistema, así como una guía para comprender su funcionamiento y configuración.

Objetivo del Proyecto

El objetivo principal del proyecto es desarrollar un sistema robusto y escalable el cual presenta a 3 tres tipos de usuario (administrador, reclutador y candidato), este permite a los usuarios administradores y reclutadores registrar ofertas en la plataforma, editar la información de ofertas creadas y ser eliminados en caso necesario, al mismo tiempo permite a los candidatos visualizar las ofertas existentes y aplicar a estas de forma sencilla.

Además, se busca proporcionar una interfaz de programación de aplicaciones (API) REST que permita a otras aplicaciones interactuar con la base de datos de usuarios y ofertas a través de operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

Requerimientos Funcionales

El desarrollo del Sistema de Gestión de Ofertas incluye la implementación de las siguientes funcionalidades:

1. Creación y edición de ofertas de empleo:
    - Implementar un formulario simple que permita a los reclutadores publicar y editar ofertas de empleo.
    - Incluir detalles básicos como el título del puesto, descripción, ubicación, salario, y tipo de contrato.
2. Listado de ofertas para candidatos:
    - Implementar una funcionalidad que permita a los candidatos ver un listado de las ofertas de empleo disponibles.
    - El listado debe mostrar información relevante como el título del puesto, ubicación, salario, y la fecha de publicación.
3. Aplicación a ofertas de empleo:
    - Permitir a los candidatos aplicar a las ofertas de empleo enviando su nombre y correo electrónico (sin almacenamiento, solo una simulación de envío).

4. API RESTful:
    - Crear una API RESTful básica que permita la creación, lectura, actualización y eliminación de ofertas de empleo.

Requerimientos Técnicos

El desarrollo del Sistema de Gestión de Ofertasse basa en los siguientes requerimientos técnicos:

1. **ASP.NET MVC:** Utilización del framework ASP.NET MVC para la implementación de las páginas web del sistema.
2. **ASP.NET Web API:** Implementación de una API REST utilizando ASP.NET Web API para permitir la interacción con la base de datos de ofertas.
3. **SQL Server y ORM:** Utilización de SQL Server para el almacenamiento de la información de las ofertas en base de datos y la configuración de un ORM (Object Relational Mapper) para facilitar las operaciones de persistencia de datos.
4. **Patrones de Diseño:** Aplicación de al menos un patrón de diseño en la implementación del sistema para mejorar su mantenibilidad y escalabilidad.

Decisiones de diseño

El proyecto se desarrolló en base a la siguiente configuración:

Framework: El proyecto se desarrolló utilizando el framework ASP.NET MVC en la versión .NET 8.0 como el framework principal.

Motor de Base de Datos: El proyecto utiliza SQL Server 2022 como el motor de base de datos para almacenar la información tanto de las ofertas como de los usuarios.

ORM: El proyecto utiliza como ORM (Object Relational Mapper) a Entity Framework Core para el mapeo y la interacción del proyecto con la base de datos SQL Server y realizar las operaciones CRUD en la tabla de ofertas. Además, se implementa una extensión mediante una **API REST** para la gestión de la tabla de usuarios. Este se seleccionó por su facilidad de uso y su compatibilidad con el Framework .NET.

- **Versión Específica:** Entity Framework Core 8.0.4

Paquetes NuGet adicionales: Además del paquete de Entity Framework Core, se utilizó un paquete adicional relacionado a Entity Framework.

- **Microsoft.EntityFrameworkCore.SqlServer 8.0.4:** Esta es una biblioteca proporciona el proveedor de SQL Server para Entity Framework Core, la cual permite la comunicación con la base de datos SQL Server.

Arquitectura: Para el desarrollo del Sistema de Gestión de Ofertas se implementó el patrón de arquitectura Modelo-Vista-Controlador (MVC). Este patrón permite separar las responsabilidades de la aplicación en tres componentes o capas principales:

- **Modelo (Model):** Es la capa encargada de representar los datos y la lógica del negocio, en esta capa se definen las clases que representan las entidades y las operaciones que se pueden realizar sobre ellas.

- **Vista (View):** Es la capa encargada de mostrar la interfaz de usuario al usuario final, en esta capa se definen las páginas web, las plantillas HTML y los elementos de la interfaz gráfica que el usuario puede interactuar.
- **Controlador (Controller):** Es la capa encargada de gestionar las solicitudes del usuario y coordinar la interacción entre el Modelo y la Vista. En esta capa se definen los métodos que responden a las solicitudes HTTP, procesan la información recibida, interactúan con el Modelo y seleccionan la Vista adecuada para mostrar al usuario.

Patrones de Diseño: Dejando de lado el patrón de arquitectura MVC, dentro del proyecto se implementaron los siguientes patrones de diseño con el fin de mejorar la eficiencia y eficacia del mismo:

- **Repositorio (Repository):** Este patrón se utiliza para separar la lógica de negocio de la lógica de acceso a datos en la aplicación, de modo que actúa como una capa intermedia entre la lógica de la aplicación y el almacenamiento de datos. Este patrón genera una abstracción del acceso a datos lo cual facilita la interacción a través de los repositorios con métodos mas simples.
- **DTO (Data Transfer Object):** Este patrón se utiliza para transferir datos entre las capas de la aplicación de manera estructurada y eficiente, funcionando de la misma manera entre el servidor y el cliente para el caso del API REST, también permite separar o desacoplar las capas de la aplicación de modo que los cambios que se realicen en la estructura de las entidades u objetos de dominio no afectaran la capa ni la lógica de presentación. Además, este patrón permite controlar que datos se exponen y como se expondrán en la capa de presentación, proporcionando una mayor seguridad y control sobre los mismos, también ayudan a reducir la carga en el servidor al evitar el procesamiento de datos innecesarios que podrían estar presentes en los objetos de dominio.
- **Inyección de dependencias:** Este patrón se utiliza para lograr un acoplamiento más bajo entre las clases y facilitar el modularidad, la reutilización y las pruebas unitarias en el proyecto. Este patrón permite desacoplar los componentes eliminando las dependencias directas entre componentes facilitando las modificaciones y reutilización de los componentes individualmente, lo que hace que estas sean más flexibles y fáciles de mantener.

API REST: Se desarrolló una API REST utilizando el framework ASP.NET Web API para realizar operaciones CRUD sobre las bases de datos de ofertas y usuarios. La gestión de usuarios se implementó únicamente a través de la API REST, debido a limitaciones de tiempo y a que no estaba contemplada dentro del alcance principal del proyecto.

Reutilización de código: El proyecto de Web API se desarrolló utilizando el proyecto MVC como referencia y base de código existente en la misma solución, de modo que se aprovecharon las funcionalidades y estructuras existentes del proyecto MVC para agilizar el desarrollo del proyecto Web API, de modo que, se reutilizaron ciertos componentes, como modelos de datos, lógica de negocio y validaciones (Modelo, Repositorios, Interfaces, contexto de conexión a base de datos), del proyecto MVC en el proyecto Web API para mantener consistencia y evitar duplicación de código. Esto con el fin de no duplicar código que puede ser reutilizado fácilmente dentro del proyecto de Web API.

Creación de la base de datos

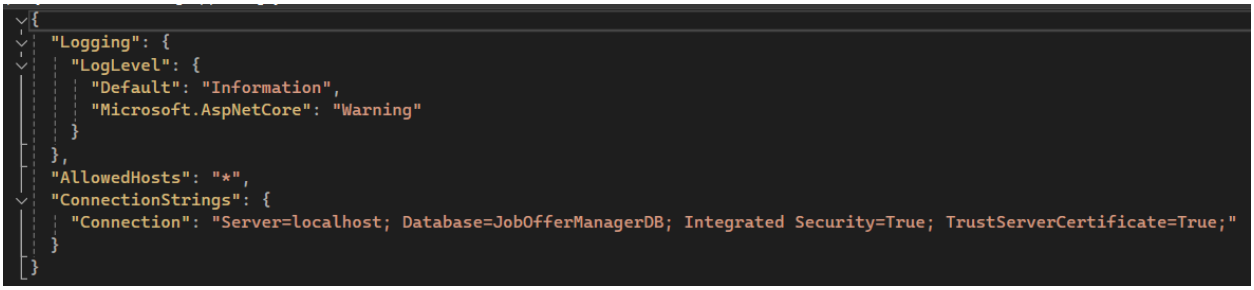
Para la creación de la Base de datos se deben ejecutar se debe utilizar el script asociado en el proyecto, el cual contiene todas las tablas necesarias, con sus respectivas relaciones e información de inicio necesaria para iniciar directamente a la gestión de ofertas en SQL Server.

- **Script de creación de BD: "**Creación BD sistema de gestión de ofertas.sql" https://github.com/JhonBulla/SistemaGestionOfertas/blob/master/Scripts/Creacion%20BD%20sistema%20de%20gestion%20de%20ofertas.sql

Configuración del ORM (Entity Framework)

Para el Sistema de Gestión de Usuarios se Entity Framework Core 8.0.4 como ORM para la interacción con la base de datos utilizando la siguiente configuración:

Cadena de conexión: Puesto que Entity Framework utiliza una cadena de conexión para establecer la conexión con la base de datos. Se especifico la cadena de conexión el archivo de configuración appsettings.json tanto del proyecto MVC como del proyecto Web API.



```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "Connection": "Server=localhost; Database=JobOfferManagerDB; Integrated Security=True; TrustServerCertificate=True;"
  }
}
```

DbContext: Para la configuración de Entity Framework se definió la clase ModelContext.cs que hereda de DbContext y define los conjuntos de entidades y las configuraciones de mapeo para la base de datos. DbContext que representa el contexto de base de datos y contiene el DbSet para las entidades necesarias para el funcionamiento del sistema de gestión, los cuales se quiere mapear de las tablas de la base de datos. Y un DbSet para las entidades correspondientes.

```
/// <summary> Conjunto de entidades de usu en la base de datos.
7 referencias
public virtual DbSet<User> Users { get; set; }

/// <summary> Representa el conjunto de entidades de tipo UserDto en el contexto ...
0 referencias
public DbSet<SistemaGestionOfertas.Models.DTO.UserDto> UserDto { get; set; } = default!;

/// <summary> Conjunto de entidades de paises en la base de datos.
2 referencias
public DbSet<Country> Countries { get; set; }
/// <summary> Conjunto de entidades de departamentos en la base de datos.
3 referencias
public DbSet<Department> Departments { get; set; }
/// <summary> Conjunto de entidades de ciudades en la base de datos.
3 referencias
public DbSet<City> Cities { get; set; }
/// <summary> Conjunto de entidades de tipo de contrato de la oferta en la base  ...
2 referencias
public DbSet<ContractType> ContractTypes { get; set; }
/// <summary> Conjunto de entidades de tiempo de expiracion de la oferta en la b ...
2 referencias
public DbSet<ExpirationTime> ExpirationTimes { get; set; }
/// <summary> Conjunto de entidades de salarios en la base de datos.
2 referencias
public DbSet<Salary> Salaries { get; set; }

/// <summary> Conjunto de entidades de ofertas en la base de datos.
5 referencias
public virtual DbSet<JobOffer> JobOffers { get; set; }

/// <summary> Representa el conjunto de entidades de tipo JobOfferDto en el cont ...
0 referencias
public DbSet<SistemaGestionOfertas.Models.DTO.JobOfferDto> jobOfferDto { get; set; } = default!;
```

```
//Configuracion de la cadena de conexion
builder.Services.AddDbContext<ModelContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("Connection")));
```

Configuración de DBContext: Desde el archivo Program.cs, se configuró el DbContext para que pueda ser inyectado automáticamente en otras partes de tu aplicación que lo necesiten, como en tus controladores o clases de repositorio. Esto se realizó a través de la función AddDbContext, donde también se relaciona con la cadena de conexión proporcionada.