Solución al Coding Challenge: Contact Manager API

Ana Karla Caballero González

23 de mayo de 2025

1. Introducción

Este informe describe la solución desarrollada para el reto de programación propuesto, que consiste en construir una API RESTful para la gestión de contactos utilizando NET Core, C#, Entity Framework y SQL Server. El objetivo es cumplir con los requisitos funcionales y no funcionales especificados, aplicando buenas prácticas de diseño y desarrollo de software.

2. Requerimientos del Problema

- El API debe ser un proyecto ASP.NET Core Web API
- Debe utilizar Entity Framework Core con enfoque Code First.
- La base de datos debe ser SQL Server y llamarse `UserManger`.
- El API debe consumir y retornar datos en formato JSON
- Se debe utilizar JWT Bearer Token para autenticación y autorización.
- El endpoint principal debe permitir operaciones CRUD sobre contactos.
- Se deben cumplir validaciones específicas sobre los datos de los contactos.
- El endpoint DELETE debe estar restringido a administradores cubanos.
- El código debe ser testable y seguir buenas prácticas (SOLID, separación de responsabilidades, etc.).

3. Arquitectura General de la Solución

La solución se estructura en los siguientes componentes principales:

1. **Modelo de Datos**: Clases `User` y `Contact` con sus restricciones y validaciones.
2. **Capa de Acceso a Datos**: Uso de Entity Framework Core y el contexto `ContactContext`.
3. **Servicios**: Lógica de negocio encapsulada en la interfaz y clase `IContactService` y `ContactService`.
4. **Controladores**: Exponen los endpoints RESTful y delegan la lógica al servicio.
5. **Autenticación y Autorización**: Configuración de JWT y políticas de acceso.
6. **Configuración**: Uso de `appsettings.json` para cadenas de conexión y parámetros de seguridad.
7. **Validaciones**: Validaciones automáticas por atributos y validaciones de negocio en los servicios.
8. **Pruebas**: (Opcional) Proyecto de tests automatizados para los controladores y servicios.

4. Descripción de la Implementación

– **Modelo de Datos**: Se definen las entidades `User` y `Contact` con sus propiedades y restricciones.

– **Persistencia**: Se utiliza Entity Framework Core con migraciones y configuración Code First.

– **Servicios**: La lógica de negocio (validaciones, reglas de unicidad, cálculo de edad, etc.) se implementa en `ContactService`.

– **Controladores**: Los controladores reciben las peticiones, validan el modelo y delegan la lógica al servicio.

– **Autenticación**: Se configura JWT Bearer Token, validando issuer, audience y claims requeridos.

– **Autorización**: Se implementa una política para restringir el acceso al endpoint DELETE a administradores cubanos.

– **Validaciones**: Se aplican tanto a nivel de modelo (atributos) como de negocio (en el servicio).

– **Configuración**: Toda la configuración sensible se almacena en `appsettings.json`.

5. Modelo de Datos

El modelo de datos está compuesto por dos entidades principales: `User` y `Contact`. Ambas entidades se implementan como clases C# y se configuran mediante atributos de validación para garantizar la integridad de los datos.

Entidad User

La entidad `User` representa a los usuarios del sistema y contiene los siguientes campos:

– **Id**: Identificador único (GUID).

– **FirstName**: Nombre del usuario, requerido, máximo 128 caracteres.

– **LastName**: Apellido del usuario, requerido, máximo 128 caracteres.

– **Username**: Nombre de usuario, requerido, máximo 60 caracteres, único.

– **Password**: Contraseña, requerida, máximo 256 caracteres.

Entidad Contact

La entidad `Contact` representa los contactos gestionados por los usuarios y contiene los siguientes campos:

– **Id**: Identificador único (GUID).

– **FirstName**: Nombre del contacto, requerido, máximo 128 caracteres.

– **LastName**: Apellido del contacto, opcional, máximo 128 caracteres.

– **Email**: Correo electrónico, requerido, máximo 128 caracteres, formato válido, único.

– **DateOfBirth**: Fecha de nacimiento, requerida, debe ser una fecha válida.

– **Phone**: Teléfono, requerido, máximo 20 caracteres.

– **Owner**: Identificador del usuario propietario del contacto (GUID).

– **Age**: Propiedad calculada (no mapeada en base de datos) que retorna la edad del contacto.

Validaciones

Las validaciones automáticas se implementan mediante atributos de anotación en las clases del modelo, como `[Required]`, `[MaxLength]`, y `[EmailAddress]`. Además, se definen índices únicos para los campos `Username` en `User` y `Email` en `Contact` para garantizar la unicidad a nivel de base de datos.

Relaciones

Cada contacto está asociado a un usuario mediante el campo `Owner`, lo que permite identificar el propietario de cada contacto.

6. Servicios

La lógica de negocio de la aplicación se encuentra encapsulada en la capa de servicios, siguiendo el principio de separación de responsabilidades. Esta capa permite centralizar las reglas de negocio y facilita la reutilización y el testeo del código.

Interfaz IContactService

Se define la interfaz `IContactService`, que especifica los métodos necesarios para gestionar los contactos, tales como la obtención, creación, actualización y eliminación de contactos. Esto permite desacoplar la lógica de negocio de los controladores y facilita la implementación de pruebas unitarias.

Implementación ContactService

La clase `ContactService` implementa la interfaz `IContactService` y contiene la lógica para:

– Obtener la lista de contactos y contactos individuales, devolviendo siempre la edad calculada.

– Validar la unicidad del correo electrónico antes de crear o actualizar un contacto.

– Validar que el contacto tenga al menos 18 años al momento de la creación o actualización.

– Asociar el contacto al usuario autenticado mediante el campo `Owner`.

– Garantizar que el campo `Owner` no pueda ser modificado por error en una actualización.

– Eliminar contactos de forma segura, validando su existencia.

Ventajas de la Capa de Servicios

– Permite mantener los controladores ligeros y enfocados en la gestión de las peticiones HTTP

– Facilita la reutilización de la lógica de negocio en diferentes partes de la aplicación.

– Mejora la testabilidad del sistema, ya que los servicios pueden ser fácilmente simulados o probados de forma aislada.

– Favorece la aplicación de principios SOLID, especialmente la inversión de dependencias y la responsabilidad única.

7. Controladores

Los controladores son los encargados de exponer los endpoints RESTful de la API y de gestionar las peticiones HTTPĖn esta solución, el controlador principal es `ContactsController`, que se encarga de las operaciones CRUD sobre los contactos.

Responsabilidades del Controlador

– Recibir y procesar las solicitudes HTTP de los clientes.

– Validar el modelo de datos recibido mediante `ModelState`.

– Delegar la lógica de negocio a la capa de servicios, manteniendo el controlador ligero y enfocado en la gestión de la petición y la respuesta.

– Retornar los códigos de estado HTTP apropiados según el resultado de la operación (por ejemplo, 201 para creación exitosa, 400 para errores de validación, 404 para recursos no encontrados, 204 para operaciones exitosas sin contenido, etc.).

– Gestionar la autorización de los endpoints, aplicando políticas específicas como la restricción del método DELETE a administradores cubanos.

Endpoints Implementados

– **POST /api/contacts**: Crea un nuevo contacto. Valida los datos y retorna 201 o 400 según corresponda.

– **GET /api/contacts**: Devuelve la lista de contactos del usuario autenticado.

– **GET /api/contacts/{id}**: Devuelve un contacto específico por su identificador.

– **PUT /api/contacts/{id}**: Actualiza los datos de un contacto existente.

– **DELETE /api/contacts/{id}**: Elimina un contacto. Este endpoint está protegido por una política que solo permite el acceso a administradores cubanos.

Ventajas de la Estructura Adoptada

– Facilita el mantenimiento y la extensión de la API, ya que cada controlador tiene una responsabilidad clara y limitada.

– Permite una integración sencilla con la capa de servicios, promoviendo la reutilización de la lógica de negocio.

– Mejora la seguridad y el control de acceso mediante el uso de atributos de autorización y políticas personalizadas.

8. Autenticación y Autorización

La seguridad de la API se implementa mediante el uso de JWT (JSON Web Tokens) para la autenticación y autorización de los usuarios. Esta estrategia permite proteger los endpoints y controlar el acceso según los roles y claims definidos en el token.

Autenticación con JWT

- La API requiere que todas las solicitudes estén autenticadas mediante un token JWT válido.
- El token debe ser emitido por el proveedor configurado (`GSI Challenge Authenticator`) y tener como audiencia `www.gsichanllengeapi.com`.
- El claim `sub` del token contiene el nombre de usuario, que se utiliza para identificar al usuario que realiza la petición.
- El token debe incluir el claim `country` con el código ISO 2 del país desde donde se autentica el usuario.

Autorización y Políticas de Acceso

- Se define una política personalizada llamada `CubanAdminsOnly`, que restringe el acceso al endpoint DELETE únicamente a usuarios con el claim `country` igual a `CU` y el rol de `Administrator`.
- Los demás endpoints requieren únicamente que el usuario esté autenticado.
- La configuración de autenticación y autorización se realiza en el archivo `Program.cs`, donde se establecen los parámetros de validación del token y las políticas de acceso.

Ventajas de la Solución de Seguridad

- Permite un control granular sobre el acceso a los recursos de la API
- Facilita la integración con sistemas de autenticación externos y la gestión de roles y claims.
- Mejora la seguridad general de la aplicación, asegurando que solo usuarios autorizados puedan realizar operaciones sensibles.

9. Configuración

Toda la configuración sensible de la aplicación, como la cadena de conexión a la base de datos y los parámetros de seguridad para JWT, se almacena en el archivo `appsettings.json`. Esto permite separar la configuración del código fuente y facilita la gestión de diferentes entornos (desarrollo, pruebas, producción).

10. Cumplimiento de los Requisitos

- El API es un proyecto ASP.NET Core Web API
- Se utiliza Entity Framework Core con enfoque Code First.
- La base de datos es SQL Server y se configura en `appsettings.json`.
- El API consume y retorna datos en JSON
- Se implementa autenticación y autorización con JWT
- El endpoint principal permite operaciones CRUD sobre contactos.
- Se aplican todas las validaciones requeridas.
- El endpoint DELETE está protegido por una política de autorización.
- El código es testable

11. Conclusión

La solución desarrollada cumple con todos los requisitos funcionales y técnicos planteados en el reto. Se ha implementado una arquitectura limpia y escalable, aplicando principios SOLID y buenas prácticas de desarrollo. La API es segura, testable y fácil de mantener, lo que facilita su evolución futura y su integración con otros sistemas.