Proyecto para el curso RESTful Web APIs con .NET Core

Kevin Andrés Hernández Rostrán[1]

[1]Universidad Cenfotec, San José, San Pedro, Costa Rica

1 Introducción

A través de este proyecto, los estudiantes pondrán en práctica los conocimientos adquiridos en el curso de desarrollo de APIs RESTful con ASP.NET Core. Se creará un servicio web que servirá como punto de partida para desarrollar aplicaciones más complejas y sofisticadas.

1.1 Objetivos Básicos

- **Crear una API RESTful básica:** Implementar los métodos HTTP fundamentales (GET, POST, PUT, DELETE) para realizar operaciones CRUD sobre un recurso sencillo (por ejemplo, una lista de tareas, una colección de usuarios).
- **Utilizar ASP.NET Core:** Configurar un proyecto de ASP.NET Core, crear controladores, y modelar datos utilizando C#.
- **Entender el ciclo de vida de una solicitud HTTP:** Desde que se recibe una solicitud hasta que se envía una respuesta.
- **Implementar enrutamiento básico:** Configurar rutas para diferentes operaciones sobre los recursos.

1.2 Objetivos Intermedios

- **Implementar validación de datos:** Asegurarse de que los datos recibidos en las solicitudes sean válidos y consistentes.
- **Manejar errores:** Implementar mecanismos para manejar errores y devolver respuestas informativas.
- **Utilizar un ORM:** Interactuar con una base de datos utilizando un ORM como Entity Framework Core.
- **Implementar autenticación básica:** Proteger la API utilizando mecanismos de autenticación sencillos como Basic Authentication o token-based authentication.

1.3 Objetivos Avanzados

- **Implementar autorización:** Controlar el acceso a los recursos de la API en función de los roles de los usuarios.
- **Utilizar middleware:** Crear middleware personalizado para realizar tareas comunes como la autenticación, la autorización, el logging o la compresión.
- **Implementar paginación:** Manejar grandes conjuntos de datos de forma eficiente.
- **Utilizar Swagger o OpenAPI:** Generar documentación interactiva de la API.

A continuación, se presentan varias propuestas de proyectos para que el estudiante seleccione la que más se adapte a sus intereses y objetivos de aprendizaje. Cada proyecto ofrece una oportunidad única para aplicar los conocimientos adquiridos en el curso de una manera práctica y creativa.

2 Propuesta de proyecto A: Reverse Proxy con API de Hacienda

2.1 Objetivo

Desarrollar un reverse proxy utilizando ASP.NET Core que actúe como intermediario entre una aplicación cliente y la API de Hacienda de Costa Rica. Este proyecto permitirá a los estudiantes aplicar los conocimientos adquiridos en el curso para

construir una solución que mejore el rendimiento, la seguridad y la escalabilidad de las aplicaciones que interactúan con la API de Hacienda.

2.2 Requisitos Mínimos

- **Configuración del proxy:** Configurar el proxy para enrutar las solicitudes a los endpoints correspondientes de la API de Hacienda.
- **Cacheo de respuestas:** Implementar un mecanismo de cacheo para almacenar las respuestas más comunes de la API y mejorar el rendimiento.
- **Manejo de errores:** Implementar un manejo de errores robusto para capturar y gestionar las excepciones que puedan ocurrir durante las solicitudes.
- **Autenticación:** Implementar un mecanismo de autenticación para acceder a la API de Hacienda de forma segura.

2.3 Temas a Abordar

- Aplicar los conceptos básicos de HTTP y desarrollo de APIs web para comprender el funcionamiento del proxy.
- Utilizar los principios de REST para diseñar la arquitectura del proxy y las interacciones con la API de Hacienda.
- Utilizar JSON para representar las respuestas de la API y para almacenar los datos en caché.
- Implementar middleware para manejar la autenticación, el cacheo y el manejo de errores.
- Diseñar una estructura de URL clara y consistente para los endpoints del proxy.
- Generar una documentación completa de la API del proxy utilizando Swagger o OpenAPI.

2.4 Requisitos Adicionales (Opcionales)

- **Compresión de respuestas:** Implementar la compresión de las respuestas para reducir el tamaño de los datos transmitidos.
- **Seguridad:** Implementar medidas de seguridad adicionales, como el cifrado de las comunicaciones y la protección contra ataques comunes.
- **Monitoreo:** Implementar un sistema de monitoreo para rastrear el rendimiento del proxy y detectar problemas.

2.5 Tecnologías Sugeridas

- **.NET Core:** Framework para el desarrollo del proxy.
- **HttpClient:** Para realizar las solicitudes a la API de Hacienda.
- **Redis:** Para implementar el cacheo.
- **Swagger/OpenAPI:** Para generar la documentación de la API del proxy.

2.6 Entrega

- Código fuente del proyecto.
- Documentación de la API del proxy en formato Swagger o OpenAPI.
- Un informe que describa la arquitectura del proxy, las decisiones de diseño y los desafíos enfrentados.
- Diagrama de la arquitectura, mostrando los componentes principales: proxy, servicios externos, base de datos (opcional).

2.7 Evaluación

- Corrección de la implementación del proxy.
- Eficiencia del cacheo y el manejo de errores.
- Seguridad de la implementación.
- Calidad del código y la documentación.

2.8 Consideraciones Adicionales

- **Documentación de la API de Hacienda:** Es fundamental que los estudiantes consulten la documentación oficial de la API de Hacienda para entender los endpoints disponibles, los formatos de datos y los requisitos de autenticación.
- **Performance:** Se debe prestar atención al rendimiento del proxy, especialmente en términos de tiempo de respuesta y uso de recursos.
- **Escalabilidad:** El proxy debe ser diseñado para poder escalar horizontalmente si es necesario.

2.9 Referencias

- API Ministerio de Hacienda.
- Hecho en Costa Rica: Hacienda CLI.

Al finalizar este proyecto, los estudiantes habrán adquirido habilidades prácticas en el desarrollo de proxies, la integración con APIs externas y la aplicación de los conceptos aprendidos en el curso.

3 Propuesta de proyecto B: API para Factura Electrónica

3.1 Objetivo

Desarrollar una API RESTful utilizando ASP.NET Core para gestionar la emisión de facturas electrónicas. Esta API permitirá a las empresas generar, consultar y cancelar facturas electrónicas de acuerdo con los estándares establecidos por el Ministerio de Hacienda de Costa Rica. El proyecto permitirá a los estudiantes aplicar los conocimientos adquiridos en el curso para construir una solución que automatice y optimice los procesos de facturación electrónica.

3.2 Requisitos Mínimos

- **Modelos de datos:** Definir modelos para representar facturas (número, fecha de emisión, cliente, productos, impuestos), clientes, productos y otros elementos relevantes.
- **Endpoints:** Implementar endpoints RESTful para las siguientes operaciones:
  - **Facturas:** Crear, leer, actualizar, cancelar facturas.
  - **Clientes:** Crear, leer, actualizar, eliminar clientes.
  - **Productos:** Crear, leer, actualizar, eliminar productos.
  - **Generación de XML:** Generar el archivo XML de la factura con el formato requerido por el Ministerio de Hacienda.
  - **Firma digital:** Implementar la firma digital de la factura utilizando un certificado digital.
  - **Envío al Ministerio de Hacienda:** Enviar la factura firmada al sistema del Ministerio de Hacienda.
- **Validación de datos:** Implementar una rigurosa validación de los datos de entrada para asegurar la integridad de las facturas.
- **Manejo de errores:** Implementar un mecanismo de manejo de errores personalizado para devolver mensajes de error claros y concisos.

- **Documentación:** Generar documentación de la API utilizando herramientas como Swagger o OpenAPI.

3.3 Temas a Abordar

- Aplicar los conceptos básicos de HTTP y desarrollo de APIs web para diseñar la arquitectura de la API.
- Utilizar los principios de REST para diseñar los endpoints y las respuestas de la API.
- Utilizar JSON para representar los datos y definir los formatos de respuesta.
- Implementar middleware para tareas como la validación de datos, la autenticación, la autorización y el manejo de excepciones.
- Diseñar una estructura de URL clara y consistente para los endpoints.
- Generar una documentación completa de la API utilizando Swagger o OpenAPI.
- Implementar un mecanismo de autenticación para proteger el acceso a la API.

3.4 Requisitos Adicionales (Opcionales)

- **Generación de reportes:** Generar reportes de ventas y otros informes a partir de los datos de las facturas.
- **Notificaciones:** Enviar notificaciones por correo electrónico o SMS cuando se emiten o cancelan facturas.

3.5 Tecnologías Sugeridas

- **.NET Core:** Framework para el desarrollo de aplicaciones web.
- **Entity Framework Core:** ORM para interactuar con la base de datos.
- **Swagger/OpenAPI:** Para generar la documentación de la API.
- **Librería de firma digital:** Para firmar los archivos XML de las facturas.
- **Biblioteca para consumir la API del Ministerio de Hacienda:** Para enviar las facturas al sistema del Ministerio.

3.6 Entrega

- Código fuente del proyecto.
- Documentación de la API en formato Swagger o OpenAPI.
- Un documento explicativo que describa las decisiones de diseño y las tecnologías utilizadas.
- Diagrama de la arquitectura, mostrando los componentes principales: api, servicios externos, base de datos (opcional).

3.7 Evaluación

- Corrección de la implementación de los endpoints.
- Calidad del código (legibilidad, mantenibilidad).
- Completitud de la documentación.
- Cumplimiento de los requisitos adicionales.
- Cumplimiento de los estándares de facturación electrónica del Ministerio de Hacienda.

3.8 Consideraciones Adicionales

- **Legislación:** Es fundamental que los estudiantes se aseguren de cumplir con toda la legislación vigente en materia de facturación electrónica.
- **Seguridad:** Se deben implementar medidas de seguridad robustas para proteger la información confidencial de los clientes y de la empresa.

- **Performance:** La API debe ser eficiente y capaz de manejar un gran volumen de transacciones.

3.9 Referencias

- API libre para Factura Electrónica en Costa Rica.
- Hecho en Costa Rica: Hacienda CLI.
- Diagrama de flujo Factura Electrónica Costa Rica.

Al finalizar este proyecto, los estudiantes habrán adquirido una sólida base en el desarrollo de APIs RESTful para aplicaciones empresariales y estarán preparados para abordar proyectos más complejos en el ámbito de la facturación electrónica.

4 Propuesta de proyecto C: Firmador Electrónico

4.1 Objetivo del Proyecto

Desarrollar una aplicación que permita firmar digitalmente documentos electrónicos de forma segura y confiable. El firmador electrónico garantizará la autenticidad, integridad y no repudio de los documentos firmados.

4.2 Lo que aprenderán los estudiantes

- **Criptografía:** Algoritmos de firma digital (RSA, DSA), generación de claves, certificados digitales.
- **Seguridad de la información:** Protección de claves privadas, prevención de ataques.
- **Desarrollo de aplicaciones:** Creación de interfaces de usuario intuitivas, manejo de archivos.
- **Legislación:** Normativa relacionada con la firma electrónica.

4.3 Criterios de Aceptación

- **Funcionalidad:**
  - Firmar digitalmente diferentes tipos de archivos (XML, etc.).
  - Verificar la integridad de una firma digital.
  - Generar certificados digitales (opcional).
- **Seguridad:**
  - Almacenar las claves privadas de forma segura.
  - Proteger contra ataques como la falsificación de firmas.
  - Cumplir con los estándares de firma digital.
- **Usabilidad:**
  - Interfaz de usuario intuitiva y fácil de usar.
- **Integración:**
  - Posibilidad de integrarse con otros sistemas (opcional).

4.4 Descripción Detallada

El firmador electrónico se construirá utilizando ASP .NET Core y se desplegará en un ambiente local (no es necesario hostearlo en una nube).

4.5 Retos y consideraciones

- **Seguridad:** La seguridad de las claves privadas es fundamental.
- **Legislación:** Es necesario cumplir con la legislación vigente en materia de firma electrónica.
- **Interoperabilidad:** El firmador debe ser compatible con diferentes formatos de archivo y sistemas operativos.

4.6 Temas a Abordar

- Aplicar los conceptos básicos de HTTP para entender la comunicación entre el cliente y el servidor.
- Utilizar los principios de REST para diseñar una API que permita firmar documentos de forma remota (opcional).
- Utilizar JSON para representar los datos de la firma y los metadatos del documento.
- Implementar middleware para manejar la autenticación, la autorización y el manejo de excepciones.
- Diseñar una estructura de URL clara y consistente para los endpoints de la API (opcional).
- Generar una documentación completa de la API utilizando Swagger o OpenAPI (opcional).
- Implementar un mecanismo de autenticación robusto para proteger el acceso a las funcionalidades de firma.
- Realizar pruebas exhaustivas para garantizar la seguridad y la funcionalidad del firmador.

4.7 Aspectos adicionales a considerar

- **Tipos de firma:** Firma simple, firma cualificada, firma avanzada.
- **Integración con sistemas de gestión documental:** Para automatizar procesos de firma.
- **Firma remota:** Permitir la firma de documentos de forma remota.

4.8 Entrega

- Código fuente del proyecto.
- Documentación de la API en formato Swagger o OpenAPI.
- Un documento explicativo que describa las decisiones de diseño y las tecnologías utilizadas.
- Diagrama de la arquitectura, mostrando los componentes principales: interfaz de usuario, motor de firma, almacenamiento de claves, etc.

4.9 Referencias

- API libre para Factura Electrónica en Costa Rica.
- Hecho en Costa Rica: Hacienda CLI.
- Diagrama de flujo Factura Electrónica Costa Rica.
- xades-signer-cr.

Al finalizar este proyecto, los estudiantes habrán adquirido una sólida base en el desarrollo de APIs RESTful para aplicaciones empresariales y estarán preparados para abordar proyectos más complejos en el ámbito de la facturación electrónica.

5 Propuesta de proyecto C: Reverse Proxy de Otras APIs

5.1 Objetivo del Proyecto

Desarrollar un reverse proxy utilizando ASP.NET Core que actúe como intermediario entre una aplicación cliente y múltiples APIs externas. Este proyecto permitirá a los estudiantes aplicar los conocimientos adquiridos en el curso para construir una

solución que consolide y unifique el acceso a diferentes servicios web, mejorando la experiencia del usuario y la mantenibilidad de las aplicaciones.

5.2 Lo que aprenderán los estudiantes

- **Arquitectura de microservicios:** Cómo un proxy puede servir como puerta de entrada a múltiples servicios.
- **Manejo de solicitudes HTTP:** Enrutamiento, transformación y composición de solicitudes.
- **Caching:** Optimización del rendimiento a través del almacenamiento en caché de respuestas.
- **Resiliencia:** Manejo de errores, timeouts y reintentos.
- **Seguridad:** Autenticación, autorización y protección contra ataques comunes.

5.3 Criterios de Aceptación

- **Funcionalidad:**
  - Enrutar solicitudes a diferentes APIs basadas en la ruta de la URL.
  - Agregar encabezados y parámetros de consulta a las solicitudes.
  - Combinar datos de múltiples APIs en una sola respuesta.
- **Performance:**
  - Implementar un mecanismo de caching para mejorar la velocidad de respuesta.
  - Manejar grandes volúmenes de tráfico.
- **Seguridad:**
  - Proteger contra ataques como inyección SQL, XSS y CSRF.
  - Implementar autenticación y autorización para acceder a las APIs.
- **Escalabilidad:**
  - Diseñar el proxy para que pueda escalar horizontalmente.

5.4 Descripción Detallada

El reverse proxy se construirá utilizando ASP.NET Core y se desplegará en un ambiente local (no es necesario hostearlo en una nube).

5.5 Tecnologías sugeridas

- **.NET Core:** Framework para el desarrollo del proxy.
- **HttpClient:** Para realizar las solicitudes a las APIs externas.
- **Redis:** Para implementar el caching (opcional).
- **Swagger/OpenAPI:** Para generar la documentación de la API del proxy (opcional).

5.6 Temas a Abordar

- Aplicar los conceptos básicos de HTTP para entender el funcionamiento del proxy.
- Utilizar los principios de REST para diseñar los endpoints del proxy y las interacciones con las APIs externas.
- Utilizar JSON para representar las respuestas de las APIs y para almacenar los datos en caché.
- Implementar middleware para manejar la autenticación, el cacheo y el manejo de errores.
- Diseñar una estructura de URL clara y consistente para los endpoints del proxy.

- Generar una documentación completa de la API del proxy utilizando Swagger o OpenAPI.
- Considerar el uso de Docker para contenerizar el proxy y facilitar su despliegue.

5.7 Aspectos adicionales a considerar

- **Seguridad:** Implementar medidas de seguridad adicionales, como el cifrado de las comunicaciones y la protección contra ataques comunes.
- **Monitoreo:** Implementar un sistema de monitoreo para rastrear el rendimiento del proxy y detectar problemas.

5.8 Referencias

- Building a Reverse Proxy in .NET Core.
- Hecho en Costa Rica: Hacienda CLI.
- How to securely reverse-proxy ASP.NET Core web apps.

Al finalizar este proyecto, los estudiantes habrán adquirido habilidades prácticas en el desarrollo de proxies, la integración con APIs externas y la aplicación de los conceptos aprendidos en el curso.