Innovación Educativa en el Desarrollo de Software: Enseñanza de APIs REST mediante Design Thinking

MPEDT Hilda Díaz Rincón [a], *Dr. José Antonio Navarrete Prieto* [b], *Dr. Eric Hernández Castillo* [c], *Manuel Díaz Rojo* [d]

[a] Tecnológico Nacional de México / Instituto Tecnológico de Tlalnepantla
[b] Tecnológico Nacional de México / Instituto Tecnológico de Tlalnepantla
[c] Tecnológico Nacional de México / Instituto Tecnológico de Tlalnepantla
[d] Tecnológico Nacional de México / Instituto Tecnológico de Tlalnepantla

Resumen

La creciente problemática de vulnerabilidades en **APIs REST (Representational State Transfer)** hace imprescindible formar a estudiantes en el diseño de soluciones escalables y seguras para mitigar riesgos en servicios web. Durante el período **2024-2**, 31 estudiantes de la asignatura **Gestión de la Seguridad de la Información** participaron en una actividad educativa cuyo objetivo fue desarrollar un **prototipo de monitoreo automatizado**, aplicando conceptos avanzados en seguridad informática.

El objetivo principal del proyecto fue **fortalecer competencias prácticas** en monitoreo y protección de APIs REST mediante un enfoque basado en **aprendizaje activo**. La **justificación** radica en la necesidad de preparar a futuros profesionales capaces de enfrentar desafíos en la gestión de riesgos en sistemas distribuidos. A través de la metodología **Design Thinking**, se fomentó la creatividad y la resolución iterativa de problemas. Se integraron tecnologías como **SQL Server, Hangfire, Visual Studio 2022, .NET Core 9 con C#, WebAPI y Swagger**. Los **resultados** reflejaron un alto nivel de comprensión por parte de los estudiantes sobre monitoreo y desarrollo seguro, a pesar de ser un ejercicio experimental.

La actividad conectó teoría con práctica, destacando la relevancia de integrar investigación y aplicación técnica en un contexto educativo para abordar desafíos contemporáneos en ciberseguridad.

Palabras clave—APIs REST, aprendizaje activo, prototipo, riesgos, seguridad informática

G

Abstract

The growing problem of vulnerabilities in REST (Representational State Transfer) APIs makes it essential to train students in the design of scalable and secure solutions to mitigate risks in web services. During the 2024-2 period, 31 students of the Information Security Management course participated in an educational activity whose objective was to develop an automated monitoring prototype, applying advanced concepts in computer security.

The main objective of the project was to strengthen practical skills in monitoring and protection of REST APIs through an active learning approach. The justification lies in the need to prepare future professionals capable of facing challenges in risk management in distributed systems. Through the Design Thinking methodology, creativity and iterative problem solving were encouraged. Technologies such as SQL Server, Hangfire, Visual Studio 2022, .NET Core 9 with C#, WebAPI and Swagger were integrated. The results reflected a high level of student understanding of monitoring and safe development, despite being an experimental exercise.

The activity connected theory with practice, highlighting the relevance of integrating research and technical application in an educational context to address contemporary cybersecurity challenges.

Keywords*— APIs REST, active learning, prototyping, risks, IT security*

1. INTRODUCCIÓN

En el panorama actual de la ciberseguridad, las APIs REST (Interfaces de Programación de Aplicaciones Representacionales) son fundamentales para la comunicación entre prototipos y aplicaciones. Sin embargo, esta funcionalidad las convierte en un objetivo prioritario para los atacantes. Según el informe de IBM Security X-Force Threat Intelligence (2023), el 41% de las violaciones de datos se originan en vulnerabilidades de APIs mal gestionadas [1]. Asimismo, Gartner proyecta que, para 2025, el 90% de los ataques a aplicaciones web estarán dirigidos a APIs [2]. Estos datos evidencian la necesidad de implementar enfoques innovadores para fortalecer la seguridad en este ámbito.

La seguridad informática es una disciplina que abarca prácticas, herramientas y tecnologías diseñadas para proteger la información y los sistemas contra accesos no autorizados, alteraciones y destrucción. Un sistema de seguridad robusto debe integrar tres pilares fundamentales: confidencialidad, integridad y disponibilidad [3]. Las APIs, como componentes críticos en la interconexión de sistemas, representan un desafío particular, ya que permiten el acceso a datos y funcionalidades sensibles a través de redes.

La creciente adopción de APIs REST como núcleo de comunicación en sistemas modernos ha incrementado su exposición a amenazas de seguridad. Según OWASP, las vulnerabilidades más comunes en APIs incluyen la exposición de datos sensibles, la falta de validación de entrada y el abuso de endpoints [4]. Aunque existen soluciones de monitoreo de APIs, muchas están diseñadas para grandes corporaciones, lo que limita su adopción en organizaciones pequeñas o medianas debido a los altos costos y la complejidad de implementación. En este contexto, surge la necesidad de una solución accesible, escalable y personalizable, capaz de detectar patrones sospechosos y responder proactivamente ante amenazas.

El aprendizaje activo es un enfoque pedagógico que involucra a los estudiantes en el proceso de aprendizaje a través de actividades prácticas, reflexión y resolución de problemas. Este enfoque fomenta la comprensión profunda, el pensamiento crítico y el desarrollo de habilidades aplicadas. Según Bonwell y Eison, el aprendizaje activo implica actividades que obligan a los estudiantes a reflexionar sobre lo que están haciendo, en contraste con la enseñanza pasiva donde solo reciben información [5]. Este enfoque promueve

el compromiso intelectual y práctico, ayudando a los estudiantes a aplicar conceptos en situaciones reales.

En el ámbito educativo, el aprendizaje activo se alinea con la enseñanza de habilidades técnicas y de resolución de problemas, especialmente en áreas como la ciberseguridad. Prince enfatiza que este enfoque mejora la retención del conocimiento y la transferencia de habilidades a contextos reales, lo que es crucial en disciplinas prácticas como la seguridad informática [6]. Además, el Aprendizaje Basado en Problemas (ABP), una modalidad de aprendizaje activo, coloca a los estudiantes en el centro de su proceso educativo, permitiéndoles abordar problemas reales o simulados. Según Barrows, el ABP fomenta el pensamiento crítico, la colaboración y la aplicación práctica del conocimiento [7].

Existen soluciones comerciales para el monitoreo de APIs, como Datadog, New Relic y Postman API Monitoring, que ofrecen funcionalidades avanzadas, como análisis de rendimiento, alertas y monitoreo de disponibilidad. Sin embargo, estas plataformas tienen limitaciones específicas:

- **Costo elevado**: Muchas requieren licencias que pueden ser prohibitivas para pequeñas empresas.
- **Enfoque en el rendimiento**: La mayoría prioriza la optimización de latencia y tiempo de respuesta sobre la seguridad.

El problema central radica en la ausencia de soluciones accesibles y personalizables para monitorear y proteger **APIs REST** (Interfaces de Programación de Aplicaciones Representacionales) en tiempo real, especialmente en entornos educativos. La hipótesis plantea que, mediante herramientas como **SQL Server** (Servidor de Lenguaje de Consulta Estructurada), **Hangfire** (Biblioteca de programación de tareas en segundo plano para .NET), **Visual Studio 2022** (Entorno de Desarrollo Integrado), **.NET Core 9** (Plataforma de desarrollo de aplicaciones), **WebAPI** (Interfaz de Programación de Aplicaciones para la Web) y **Swagger** (Herramienta de documentación de APIs), junto con la aplicación de patrones de diseño, programación orientada a objetos y arquitectura **NCapas** (Arquitectura en múltiples capas), es posible desarrollar un prototipo efectivo capaz de detectar patrones sospechosos y generar alertas automáticas. Este enfoque aborda no solo un problema técnico, sino que también fomenta el aprendizaje práctico en ciberseguridad.

Para abordar esta problemática, se desarrolló un prototipo de monitoreo como parte de una actividad educativa en la asignatura *Gestión de la Seguridad de la Información*. La metodología incluyó el diseño de un middleware para registrar solicitudes y respuestas, la integración de **SQL Server** (Servidor de Lenguaje de Consulta Estructurada) para el almacenamiento de registros y el uso de **Hangfire** (Biblioteca de programación de tareas en segundo plano para .NET) para programar tareas automatizadas de análisis. **Swagger** (Herramienta de documentación de APIs) facilitó la documentación y prueba de los endpoints, mientras que la aplicación de patrones de diseño y una arquitectura **NCapas** (Arquitectura en múltiples capas) aseguró escalabilidad y modularidad. Este enfoque permitió validar la efectividad del prototipo y fortalecer las competencias técnicas de los estudiantes al conectar teoría y práctica en un entorno educativo.

Los resultados muestran que el prototipo propuesto mejora la protección de datos sensibles y es accesible, escalable y adaptable a diversas necesidades organizacionales. Destaca por mitigar riesgos de manera proactiva y ser una alternativa eficiente y personalizable frente a herramientas comerciales costosas y complejas.

2. CONTENIDO

Las APIs REST son una pieza clave en la infraestructura tecnológica actual, permitiendo la interacción entre aplicaciones y servicios. Sin embargo, su mal diseño, implementación o monitoreo puede convertirlas en objetivos de ataque, lo que las posiciona como puntos críticos de vulnerabilidad.

El objetivo fue **desarrollar un prototipo de monitoreo para mejorar la seguridad informática de las APIs REST**. Esto incluyó registrar actividades, identificar patrones sospechosos en tiempo real y emitir alertas automáticas, utilizando herramientas modernas como las mencionadas en párrafos anteriores.

Con esta práctica se busca cubrir esta brecha al proponer un prototipo accesible y escalable que utilice herramientas de código abierto y ampliamente adoptadas como .NET Core, Hangfire y SQL Server.

Las APIs REST, al operar mediante solicitudes HTTP, están expuestas a ataques como la inyección SQL, la falsificación de solicitudes entre sitios (CSRF) y el abuso de endpoints.

La metodología empleada se basa en la **investigación aplicada**, definida como aquella que tiene el propósito de resolver problemas prácticos mediante el desarrollo de soluciones que pueden ser implementadas y evaluadas en contextos específicos [8]. En este caso, se diseñó e implementó un sistema de monitoreo de APIs REST como parte de una actividad educativa en la asignatura Gestión de la Seguridad de la Información. Como parte de la aplicación de un aprendizaje activo, se utilizó el **Design Thinking**, una metodología centrada en el usuario que fomenta la creatividad y la resolución de problemas de manera iterativa. Es ideal para aplicarlo en clase porque permite a los estudiantes trabajar de forma colaborativa, diseñando soluciones prácticas para problemas reales, como el monitoreo y mejora de la seguridad informática en APIs REST.

La combinación de estas permite abordar el desarrollo del prototipo de monitoreo desde dos perspectivas complementarias:

1. **Design Thinking:** Proporciona un marco iterativo y creativo para explorar, idear y probar soluciones, asegurando que el prototipo se diseñe considerando las necesidades reales de los usuarios.
2. **Metodología Aplicada:** Enfatiza la implementación práctica de soluciones para resolver problemas específicos, utilizando herramientas y tecnologías modernas.

El desarrollo del prototipo de monitoreo para APIs REST se estructuró en fases, integrando los enfoques de **Design Thinking** y **Metodología Aplicada** para garantizar un proceso creativo y técnico. Cada fase combinó objetivos claros, acciones estratégicas y resultados concretos.

Fase 1: Empatizar (Design Thinking). El objetivo inicial fue comprender las necesidades y desafíos relacionados con la seguridad en APIs REST desde la perspectiva de los usuarios, como administradores de prototipos y equipos de TI. Los estudiantes analizaron casos reales de vulnerabilidades comunes en APIs REST, basándose en reportes de OWASP (2023), y exploraron cómo estas vulnerabilidades impactan la seguridad informática. Este análisis permitió identificar las principales áreas de riesgo y los requisitos funcionales del prototipo, estableciendo una base sólida para el diseño.**Fase 2: Definir (Design Thinking).** La siguiente etapa se centró en formular el problema que guiaría el desarrollo del prototipo. A partir de la información recopilada en la fase anterior, se definió el problema central: *"Las APIs REST presentan vulnerabilidades críticas que requieren un prototipo escalable y eficiente para monitorear actividades, detectar patrones sospechosos y emitir alertas automáticas."* Este ejercicio resultó en un enfoque claro y bien delimitado que guio todas las decisiones técnicas y de diseño.

Fase 3: Diseño del Prototipo (Metodología Aplicada). El objetivo de esta fase fue traducir el problema definido en un diseño técnico detallado. Los estudiantes realizaron varias acciones clave:

- Diseñaron un middleware en **.NET Core** para capturar datos relevantes de las solicitudes HTTP.
- Configuraron **SQL Server** como base de datos para almacenar registros de manera estructurada.
- Integraron **Hangfire** para programar análisis automatizados y generar alertas en tiempo real.
- Utilizaron **Swagger** para documentar y probar los endpoints de la API.
- Aplicaron patrones de diseño y una arquitectura **NCapas** para garantizar modularidad y escalabilidad.

El resultado fue un diseño técnico robusto que cumplió con los requisitos identificados, sirviendo como base para el desarrollo del prototipo.

Fase 4: Idear y Prototipar (Design Thinking + Metodología Aplicada). En esta fase se desarrolla un prototipo funcional del prototipo. Los estudiantes trabajaron en equipos para implementar el diseño técnico previamente definido, iterando sobre el prototipo con base en pruebas y retroalimentación continua. Como resultado, lograron un prototipo básico capaz de registrar actividades de las APIs, detectar patrones sospechosos y emitir alertas automáticas, sentando las bases para su validación en entornos controlados.

Fase 5: Probar y Validar (Design Thinking + Metodología Aplicada). Esta fase se enfocó en evaluar el prototipo desarrollado en condiciones simuladas. Se diseñaron escenarios que incluían tráfico legítimo y malicioso para verificar la capacidad del prototipo de detectar anomalías y generar alertas. Los estudiantes ajustaron el prototipo según los resultados obtenidos, asegurando que cada componente cumpliera con su propósito. Como resultado, el prototipo fue validado funcionalmente y optimizado para mejorar su desempeño.

Fase 6: Implementación Final (Metodología Aplicada)
En la fase final, el objetivo fue entregar un prototipo funcional y documentado. Los estudiantes documentaron el prototipo, incluyendo instrucciones claras de instalación y uso, y reflexionaron sobre el impacto del proyecto en sus habilidades técnicas y prácticas. El resultado fue un prototipo completo, operativo y listo para ser implementado en entornos reales, demostrando su escalabilidad y eficacia en la mejora de la seguridad en APIs REST.

RESULTADOS

Los resultados obtenidos se muestran las siguientes figuras en donde se visualizan como fue desarrollado e integrado el prototipo. Primeramente, se realiza lo que corresponde a la base de datos como es Abrir SQL Server Management Studio, colocar las credenciales, y crear la base de datos con las tablas correspondientes**, ver Figura1.**

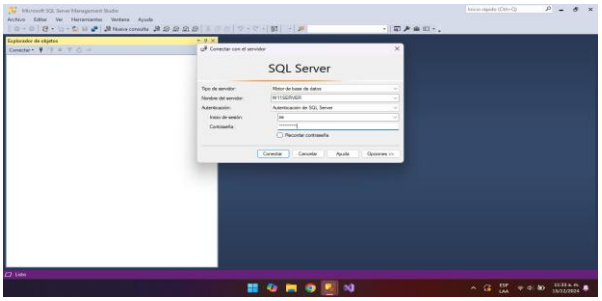

Figura 1. Apertura de SQL para la creación de la Base de Datos a utilizar. Fuente: Elaboración Propia

Posteriormente se abre Visual Studio 2022 para realizar la creación del proyecto, ver **Figura2.**

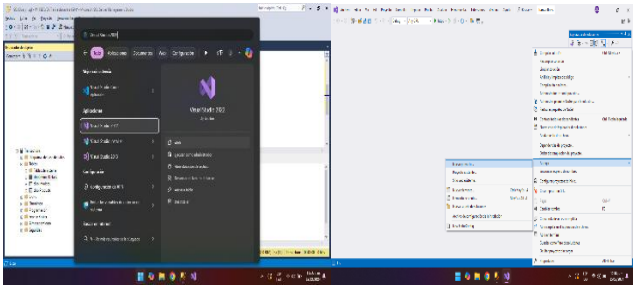

Figura 2. Creación del nuevo proyecto.
Fuente: Elaboración propia

Para garantizar una integración eficiente del proyecto, se procede con la adición de las bibliotecas y clases necesarias, asegurando su correcta funcionalidad. Como parte del proceso de optimización, se eliminan los archivos generados por **default**, permitiendo una gestión más limpia y organizada del código.

Asimismo, se instalan los paquetes esenciales que facilitarán el desarrollo y ejecución del sistema. Entre ellos, se encuentra **Hangfire**, utilizado para la gestión de tareas en segundo plano; **Swashbuckle.AspNetCore.Swagger**, que permite documentar y visualizar la API REST de manera estructurada; y **System.Data.SqlClient**, biblioteca fundamental para la conexión con la base de datos.

Estos pasos garantizan una mejor organización del proyecto, optimizando su rendimiento y facilitando la implementación de nuevas funcionalidades. Para mayor referencia, consultar la **Figura 3**.

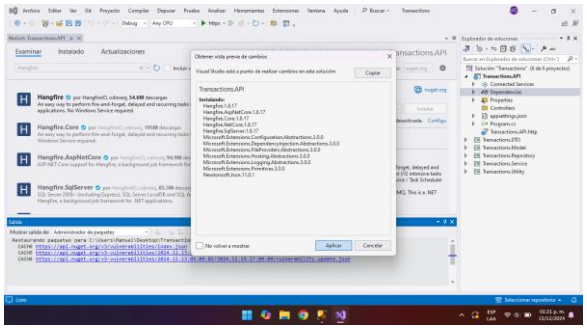

Figura 3. Instalación de Hangfire.
Fuente: Elaboración propia

Para la correcta configuración del proyecto, se **agrega la dependencia "Transactions.Service"**, permitiendo la integración de los servicios correspondientes dentro del sistema. Además, se incorpora el archivo **"launchSettings.json"**, ubicado en la carpeta **"Properties"**, con el propósito de establecer las configuraciones necesarias para que **Swagger** se configure como el proyecto de inicio, facilitando la documentación y prueba de la API.

Seguidamente, se agrega la **cadena de conexión** a la base de datos, la cual debe incluir los siguientes elementos:

Data Source=ComputerName; InitialCatalog=DataBaseName; UserID=sa; Password=passwordExample987; Connect Timeout=0; TrustServerCertificate=True;

Esta cadena de conexión es configurada de manera **global** para su acceso desde distintas capas del sistema. Posteriormente, se procede a la **configuración de Hangfire y Swagger**, asegurando su integración adecuada en el entorno de desarrollo. Además, se establecen las **tablas de Hangfire** dentro de **SQL Server**, configurándolas para que sean **agregadas automáticamente** por defecto. De este modo, en la **primera compilación y ejecución del proyecto**, las tablas serán creadas sin necesidad de intervención manual, optimizando la implementación del sistema. Para mayor referencia, consultar la **Figura 4**.

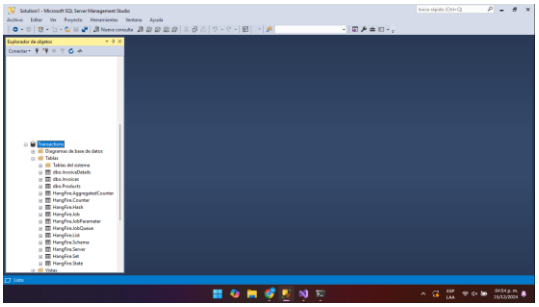

Figura 4. Tablas de Hangfire en SQL Server.
Fuente: Elaboración propia

Una vez compilada y ejecutada la solución por primera vez, se presiona clic en el panel de herramientas de Visual Studio y se ejecuta con **https://localhost:7079/swagger/index.html** , toda vez que se realiza lo anterior se visualiza el Dashboard de Hangfire en **https://localhost:7079/hangfire y se** continúa realizando la configuración Transactions.Model utilizando las herramientas y el paquete "Microsoft.EntityFrameworkCore.SqlServer", ver **Figura 5.**

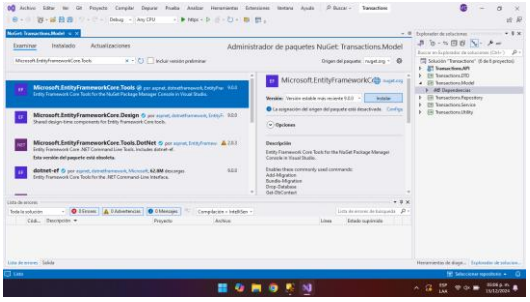

Figura 5. Instalar "Microsoft.EntityFrameworkCore.Tools".
Fuente: Elaboración propia.

En la capa de Transactions.Repository, se añade una referencia a Transactions.Model, esto debido a que esta capa actúa como intermediaria entre módulos y permite la interacción mediante abstracciones.

Se **prosigue** con la Capa **Transactions.DTO**. En esta capa estarán únicamente las clases correspondientes al **modelo de negocio**, excluyendo aquellas relacionadas con **Hangfire**, ya que los **DTO (Data Transfer Objects)** sufrirán modificaciones para evitar el uso de clases derivadas que aún no existan, ver **Figura 6.**

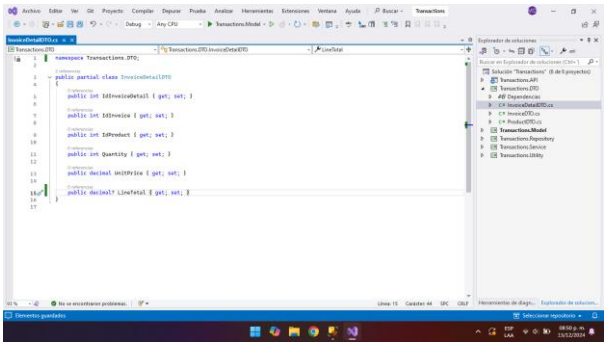

Figura 6. DTO sin propiedades de clases superiores
Fuente: Elaboración Propia

Para optimizar el manejo de solicitudes en la API, se implementan cambios que permiten la utilización de los métodos HTTP **GET, POST, PUT y DELETE** en la gestión de productos sin necesidad de interactuar con clases hijas o derivadas. Esto se logra mediante el uso de **DTO (Data Transfer Objects)**, asegurando que la creación de un producto, en el contexto de la facturación de una venta, no requiera la manipulación directa de clases dependientes.

Como parte de esta mejora, se incorpora la clase **ResponseDTO**, cuya función principal es **gestionar y monitorear** las acciones del servicio dentro del controlador. Esta clase permite **controlar el estado de cada solicitud**, proporcionando una visión clara del proceso en cada petición HTTP.

Para complementar esta implementación, se integra **Automapper**, facilitando la conversión de instancias entre **ModelDTO y Model**, lo que optimiza la manipulación de datos. En la capa **Transactions.Service**, se desarrolla la clase **ProductService.cs**, la cual incluye las siguientes funcionalidades:

- **IGenericRepository<Product>**, que abstrae las operaciones de acceso a datos, mejorando la modularidad del sistema.
- **Automapper**, encargado de la conversión automática entre DTO y Model.
- **Logger**, que permite registrar mensajes en la consola para el seguimiento de procesos.
- **Métodos CreateAsync, EditAsync, DeleteAsync, ConsultAsync y ObtainAsync**, los cuales utilizan **LINQ** en la capa **Transactions.Repository**, evitando el uso de consultas SQL sin procesar (**SQL Raw**).
- **Enqueue (capa Transactions.API)**, que gestiona la puesta en cola de solicitudes en caso de cargas elevadas, asegurando la ejecución óptima de peticiones en entornos de alta demanda.

Finalmente, se continúan las configuraciones necesarias para la correcta implementación del proyecto, garantizando un flujo de trabajo eficiente y escalable dentro del prototipo. Y se ejecuta el proyecto, como se muestra en la **Figura 7.**

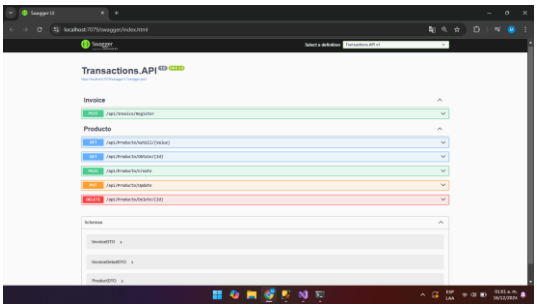

Figura 7. API de Swagger de inicio. Fuente: Elaboración propia

En la **Figura8**, se presenta la ejecución del proyecto, donde se puede observar el proceso de **vinculación con SQL** para la gestión de las peticiones realizadas. En esta visualización, se evidencia cómo la API interactúa con la base de datos, permitiendo la ejecución de solicitudes **HTTP** y garantizando el procesamiento eficiente de la información. Esta ejecución refleja el correcto funcionamiento de la integración con **SQL Server**, asegurando que cada petición enviada sea gestionada de manera adecuada y proporcionando una respuesta estructurada en función del estado de la solicitud.

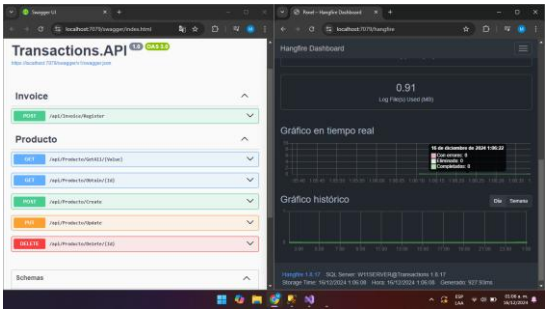

Figura 8. Dashboard vinculado con SQL Server
Fuente: Elaboración propia

En la **Figura 9**, se muestra la ejecución del registro de un movimiento en la pestaña derecha, donde se evidencia que la operación se ha realizado exitosamente. En este proceso, la clase **ResponseDTO** jugó un papel clave al capturar y mostrar la respuesta correspondiente al valor insertado.

El **Status** reflejado en la respuesta indica que la solicitud fue procesada correctamente, sin generar mensajes de error, lo que confirma que el registro se efectuó de manera exitosa. Esto valida la correcta implementación del sistema, asegurando que la inserción de datos en la base de datos se lleva a cabo sin inconvenientes y que las respuestas a las solicitudes HTTP se gestionan de manera adecuada.

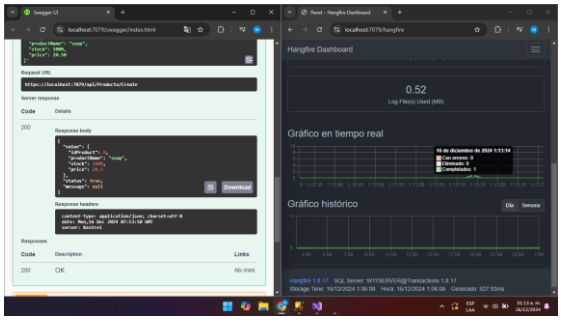

Figura 9. Petición Create registrada.
Fuente: Elaboración propia

Otra opción dentro del entorno **REST** se presenta en la **Figura 10**, donde se puede observar la **comprobación de cambios** a través de la ejecución de una solicitud **HTTP GET** utilizando el método **ObtainAsync**.

En esta visualización, se valida que la petición se procesó correctamente, confirmando que la API es capaz de recuperar los datos de manera eficiente desde la base de datos. La correcta ejecución de este método demuestra la estabilidad del sistema y la adecuada integración con el modelo de datos, garantizando que las solicitudes sean atendidas conforme a los parámetros establecidos.

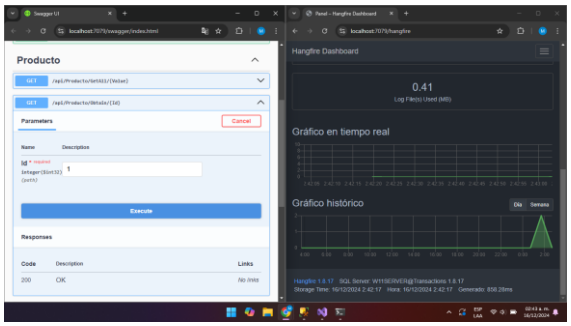

Figura10. Petición Get ejecutada.
Fuente: Elaboración propia

En la **Figura11** se puede observar cómo "Hangfire" indica que todas las tareas han sido completadas y que no existe ninguna en espera.

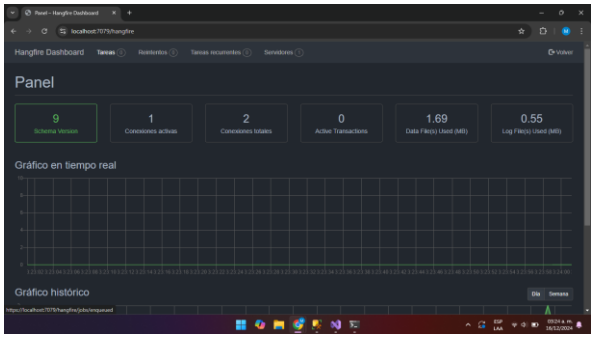

Figura 11. DashBoard de "Hangfire".
Fuente: Elaboración propia

En el DashBoard que se muestra en **la Figura 12**, se visualiza como todas las tareas que se realizaron mediante las solicitudes HTTP(post, put y delete). No se registró ninguna de lectura, esto porque no es relevante administrar los movimientos de lectura, solo se tiene que registrar los movimientos que tuvieron que ver con un cambio en los registros de la base de datos.

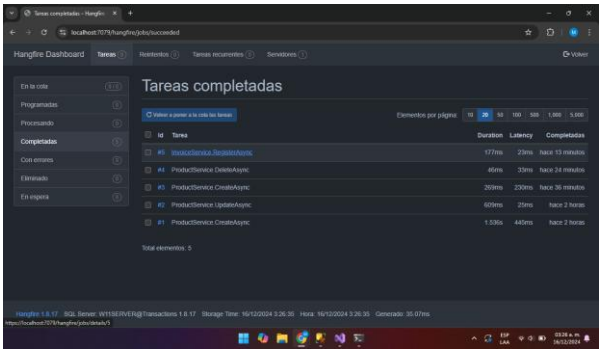

Figura 12. Jobs Completados. Fuente: Elaboración propia

Como se puede observar en la pestaña derecha se registró un movimiento, de la petición realizada. El ResponseDTO envía la respuesta del valor que se insertó, el status, y visualiza que no existe mensaje de error, esto indica que se realizó correctamente la ejecución del registro.
Además, observando dentro de la Base de Datos se comprueba el registro de JOB en SQL Server. En SQL Server realizar la consulta "SELECT * FROM Hangfire.Jobs;" mostrará los movimientos que han realizado, es decir muestra todo lo realizado en la capa DTO utilizando el controlador para la API.

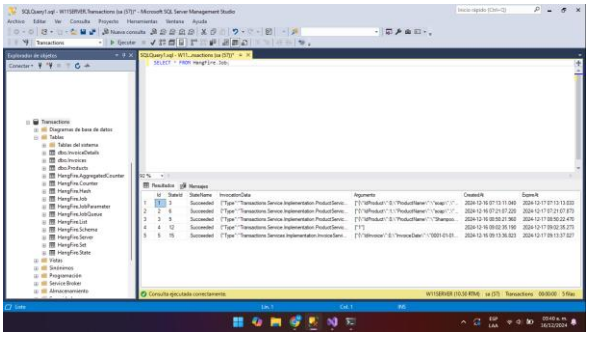

Figura 13.

Y como parte de la comprobación además se observa el registro de estado en SQL Server. Donde al realizar la consulta "SELECT * FROM HangFire.State" mostrará todavía más detalle de las colas de los trabajos. Lo más importante de observar aquí son las fases de los JOBS (Enqueued, Processing, Succeeded) ya que es donde se muestra el estado del JOB llevando su seguimiento y procedimiento mediante un ID, ver **Figura 13.**

Finalmente se observa que al desarrollar el prototipo de monitoreo de APIs REST, los estudiantes:

1. Identificaron problemas reales de vulnerabilidad en APIs REST (aprendizaje basado en problemas).
2. Diseñaron y desarrollaron soluciones prácticas utilizando herramientas modernas (aprendizaje activo aplicado).
3. Reflexionaron sobre sus resultados, mejorando su comprensión de los riesgos de seguridad y las estrategias de mitigación.

3. CONCLUSIONES Y RECOMENDACIONES

Integración Multidisciplinaria: El desarrollo del prototipo de monitoreo automatizado para APIs REST, basado en el aprendizaje activo y la metodología de Design Thinking, demostró la eficacia de un enfoque integrador que conecta múltiples asignaturas. Este proyecto permitió a los estudiantes aplicar conceptos de Metodología de la Investigación, Programación, Base de Datos, Ingeniería de Software, Arquitectura de Software, Redes y Gestión de la Seguridad de la Información, fortaleciendo su comprensión de cómo las diferentes disciplinas convergen para resolver problemas complejos de ciberseguridad.

Desarrollo de Competencias Prácticas: La implementación de herramientas como SQL Server, Hangfire, Visual Studio 2022, .NET Core 9 con C#, WebAPI y Swagger permitió a los estudiantes consolidar habilidades técnicas y críticas en

un contexto real. Esto incluyó el diseño de prototipos escalables y seguros, la programación orientada a objetos, la arquitectura en capas y la gestión de bases de datos, todo bajo la perspectiva de un problema auténtico.

Relevancia del Aprendizaje Activo y Design Thinking: El uso de Design Thinking fomentó la empatía con las necesidades del usuario, la creatividad en la generación de soluciones y la validación iterativa de prototipos. Este enfoque, combinado con el aprendizaje activo, no solo facilitó la adquisición de conocimientos técnicos, sino que también promovió el trabajo colaborativo, el pensamiento crítico y la capacidad de los estudiantes para abordar desafíos contemporáneos en ciberseguridad.

Preparación para Problemas Reales: Los estudiantes lograron conectar teoría y práctica al enfrentarse a un problema real de seguridad en APIs REST, reforzando su capacidad para diseñar e implementar soluciones técnicas aplicables en entornos laborales. Esto los posiciona como profesionales mejor preparados para los retos del mercado.

Recomendaciones

Fomentar Proyectos Integradores en el Currículo: Se recomienda incorporar proyectos similares en los planes de estudio, aprovechando su capacidad para integrar asignaturas clave. Proyectos como este deben plantearse como una actividad evaluativa transversal que permita a los estudiantes aplicar conocimientos adquiridos en diferentes etapas de su formación.

Promover el Uso de Metodologías Activas: Continuar utilizando el aprendizaje activo y el Design Thinking en actividades educativas para mejorar la participación estudiantil y la resolución de problemas. Estas metodologías deben incluir la empatía con el usuario, el prototipado y la validación como elementos centrales del proceso educativo.

Fortalecer la Interdisciplinariedad: Diseñar proyectos que involucren la colaboración entre distintas áreas de conocimiento, como programación avanzada, bases de datos y ciberseguridad, asegurando que los estudiantes entiendan la conexión entre disciplinas y cómo aplicarlas en problemas reales.

Incorporar Evaluaciones Basadas en Competencias: Implementar evaluaciones que midan el desarrollo de competencias técnicas y críticas en cada fase del proyecto. Esto incluye habilidades técnicas (desarrollo y programación), habilidades blandas (trabajo en equipo y comunicación) y la capacidad de proponer soluciones creativas.

Ampliar el Alcance a Problemas Globales: Proponer que futuros proyectos integren desafíos de ciberseguridad más amplios, como el monitoreo de entornos híbridos o la seguridad en entornos móviles. Esto ampliará las competencias de los estudiantes y los preparará para un mercado laboral en constante evolución.

Utilizar Herramientas y Tecnologías Actuales: Continuar fomentando el uso de herramientas modernas como SQL Server, Swagger y Hangfire, y actualizar las tecnologías utilizadas según las tendencias del mercado. Esto garantizará que los estudiantes trabajen con herramientas relevantes y en sintonía con la industria.

Evaluar el Impacto del Proyecto: Realizar un análisis de los resultados y habilidades adquiridas al finalizar el proyecto para identificar áreas de mejora y replicar las mejores prácticas en actividades futuras.

4. REFERENCIAS

[1] IBM Security X-Force, *IBM Security X-Force Threat Intelligence Index 2023*. IBM, 2023. [Online]. Available: https://www.ibm.com/reports/threat-intelligence/

[2] Gartner, *Predicts 2023: APIs Demand Improved Security and Management*, Gartner, 2023. [Online]. Available: https://www.gartner.com/en/documents

[3] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed. Pearson, 2017.

[4] OWASP, *OWASP API Security Top 10 - 2023*. OWASP, 2023. [Online]. Available: https://owasp.org/API-Security/

[5] C. Bonwell and J. Eison, *Active Learning: Creating Excitement in the Classroom*. Washington, D.C.: ERIC Clearinghouse on Higher Education, 1991.

[6] M. Prince, "Does Active Learning Work? A Review of the Research," *Journal of Engineering Education*, vol. 93, no. 3, pp. 223-231, 2004.

[7] H. S. Barrows, *A Taxonomy of Problem-Based Learning Methods*, Medical Education, vol. 20, no. 6, pp. 481-486, 1986.

[8] C. Hernández, R. Fernández, and P. Baptista, *Metodología de la Investigación*, 6th ed. México: McGraw-Hill, 2014.