UNIVERSIDAD TECNOLOGICA DE QUERETARO

Materia:

Seguridad en el Desarrollo de Aplicaciones.

Nombre del alumno:

Pablo Emilio Oliva Flores

Título de la actividad:

Evaluación

Nombre del docente:

M.C.C. Emmanuel Martínez Hernández

07 de Junio del 2025

Santiago de Queretaro, Qro

Contents

INTRODUCCIÓN

En la actualidad, el desarrollo de aplicaciones web con acceso a bases de datos se ha popularizado debido a sus múltiples aplicaciones en campos como el comercio electrónico, la gestión empresarial, la educación y los videojuegos, entre otros. Esta facilidad para almacenar, consultar y modificar datos eficientemente y de manera segura es uno de los pilares más importantes para el desarrollo de sistemas simples y eficientes al mismo tiempo. El presente proyecto tiene lugar entonces en este escenario; en la presente propuesta se presenta un sistema enfocado en la gestión de productos mediante un acceso web utilizando Flask, un microframework de Python conocido por su simplicidad y adecuado para cualquier aplicación; SQLite, a su vez, se utiliza como el sistema de gestión, ya que es ligera y poco pide recursos con un módulo suficientemente eficaz. El sistema final consiste en un componente de interacción entre usuario y sistema y una tabla de productos, simulando un catálogo de videojuegos; las operaciones CRUD se pueden realizar, lo que permite comprobar la eficacia del sistema propuesto.

La preocupación más importante al desarrollar interfaces web para sistemas que gestionan datos sensibles o privados radica en la seguridad del acceso a la información. Por esta razón, el proyecto incluye un sistema de autenticación basado en tokens temporales para garantizar que solo usuarios autorizados puedan realizar operaciones críticas en la base de datos. Implementamos este mecanismo utilizando la biblioteca de Python secrets y funciones de decorador que verifican este token en cada solicitud HTTP, lo que impide el acceso a rutas restringidas, lo que reduce drásticamente las vulnerabilidades comunes, como la filtración de datos debida a la exposición inadecuada. Esta práctica coincide con los protocolos de desarrollo seguro recomendados en el desarrollo web moderno y autosuficiente, lo que fortalece de manera efectiva la integridad y privacidad de la aplicación.

Sin embargo, a pesar del énfasis en la seguridad, la característica más importante que distingue al sistema es su estructura altamente modular y escalable, lo que hace que futuras extensiones o integraciones con tecnología adicional como bases de datos relacionales más complejas, frontends desarrollados en marcos más modernos como React o Angular., u otros servicios en la nube seguros y protegidos. A través de endpoints fácilmente accesibles y definidos con claridad, la API desarrollada utilizada en este proyecto es capaz de recuperar datos específicos por nombre, agregar nuevos registros al inventario, modificar un solo campo o eliminar registros, todo a través de solicitudes HTTP, mostrando un ejemplo de la extrema utilidad y seguridad de implementación de una API RESTful.

DESARROLLO

Analisis

Requerimientos Funcionales

Los requerimientos funcionales describen el comportamiento y las funcionalidades que el sistema debe ofrecer. En esta API, los principales requerimientos funcionales se agrupan por tipo de operación y se resumen de la siguiente manera:

a) Autenticación y Generación de Token

Endpoint: /token

Método: POST

Permite a un cliente autenticarse enviando un nombre y una descripción correspondientes a un producto válido. Si las credenciales coinciden con un producto registrado, el servidor genera un token hexadecimal y lo entrega al usuario junto con su tiempo de expiración.

Este token servirá para autorizar futuras peticiones a rutas protegidas.

b) Consulta de Productos

Endpoints:

/product: Retorna un producto específico por su nombre.

/product/whole: Retorna todos los productos registrados.

/product/all: Retorna productos cuyo precio al menudeo es de $1800.

Método: GET

Permite obtener información detallada de uno o más productos registrados en la base de datos.

c) Registro de Nuevos Productos

Endpoint: /product/add

Método: GET

Permite agregar un nuevo producto a la base de datos, validando que no exista previamente uno con el mismo nombre. Requiere que todos los campos estén completos.

d) Edición de Productos

Endpoint: /product/edit/<int:product_id>

Método: GET

Permite modificar un campo específico de un producto existente, siempre y cuando el campo sea válido (por ejemplo, nombre, descripción o precio).

e) Eliminación de Productos

Endpoint: /product/erase/<int:product_id>

Método: GET

Elimina un producto de la base de datos si existe. Devuelve un mensaje de confirmación o error si el ID no fue encontrado.

f) Inicio de Sesión

Endpoint: /login

Método: POST

Verifica si las credenciales enviadas (nombre y descripción) existen en la base de datos. No genera token, solo confirma autenticación simple.

g) Ruta de acceso sin autenticación

Endpoint: /admin/data

Método: GET

Simula una ruta que expone datos confidenciales, pero no requiere autenticación. Es útil para evaluar vulnerabilidades de diseño por omisión de protección.

Requerimientos de Seguridad

El análisis de seguridad contempla las medidas que aseguran la integridad, confidencialidad y autenticación de las peticiones que se hacen a la API:

a) Autenticación basada en Token

El sistema implementa tokens únicos generados mediante el módulo secrets, lo que asegura alta aleatoriedad.

Estos tokens están limitados por tiempo (TOKEN_EXPIRATION_SECONDS = 300), lo que mitiga riesgos de uso prolongado o robo de token.

b) Protección de rutas sensibles

Todas las rutas relacionadas con productos (consulta, edición, eliminación, etc.) están protegidas mediante el decorador @token_required, que valida la existencia y vigencia del token.

Este decorador actúa como un middleware de autenticación, impidiendo el acceso a usuarios no autorizados.

c) Validación de campos

En rutas como /product/add y /product/edit, se verifica que los campos requeridos estén presentes y que los campos a modificar sean válidos.

Se evita la sobreescritura de campos inesperados.

d) Prevención de duplicados

En la inserción inicial de productos (init_db) y en la ruta de inserción de nuevos productos, se realiza una validación previa para evitar que se dupliquen registros.

e) Omisiones intencionadas para análisis

La existencia de la ruta /admin/data sin protección de token demuestra cómo puede quedar expuesta información si no se asegura cada endpoint.

Investigación e implementación de: Buenas prácticas, técnicas y mecanismos de protección en el código

Desde el punto de vista de la seguridad, el desarrollo del software consiste en el cumplimiento de buenas prácticas de programación y técnicas de protección y fortalecimiento de la integridad, la confidencialidad y la disponibilidad de la información. El sistema durante su desarrollo involucró varias estrategias de seguridad desde su código fuente. Uno de los ejemplos más claros de las buenas prácticas que se implementaron correctamente es la validación del input del usuario, entender el uso correcto de las estructuras de control de flujo y la separación de responsabilidades. No solo facilita significativamente la lectura del código visible e invisible, sino que también disminuye el riesgo de fallas lógicas comunes y la introducción de dependencias de seguridad adicionales. Además, no utilizar consultas SQL directas para operaciones de gene query cumple la función de proteger la integridad de la base de datos. Inyección SQL es una de las concesiones de seguridad más comunes en las aplicaciones orientadas a bases de datos.

Con respecto al manejo de las credenciales, las técnicas aplicadas para asegurar los datos de autenticación del usuario permiten tener almacenado en una base de datos segura gracias a la posibilidad de cifrado ofrecida por bcrypt para el subsistema de las contraseñas. Es decir, estos datos no son accesibles, por lo que, aunque la base de datos se vea comprometida y consigan hacer un volcado de la misma, de todas formas, no es posible acceder a los datos sensibles. Una protección extra es el uso de tokens únicos generados desde el back al momento que abra una sesión el usuario. Estos mismos son de duración limitada y permiten al back identificar de una manera segura a una persona de entre todas las que abrieron una sesión. Además, se usan para restringir el acceso a las rutas protegidas, sin posibilidad de acceso o modificación de algún recurso sensible por parte de alguna persona no autorizada.

Además, solo las operaciones críticas, como crear, actualizar o eliminar funciones, verifican el token de manera condicional. En otras palabras, agrega control de acceso rápido. Significa que las operaciones críticas solo se pueden realizar en nombre de un usuario autenticado. Además, se ha implementado de manera tipificada la gestión de errores para no exponer la información que podría proporcionar detalles sobre cómo funciona internamente el sistema. En cambio, proporcionaría un mensaje genérico al usuario final. Finalmente, se ha adoptado una buena estructura de código para futuras adiciones y modificaciones. Se logra parte al usar funciones en las que se puede reutilizar y decoradores que se puede aplicar de manera consistente en toda la aplicación. Al hacerlo, asegura que la estructura del sistema se expanda a gran escala sin perder la seguridad de la integridad del sistema. En resumen, las acciones anteriores lo protegen de las amenazas más comunes. Se asegura de que su sistema sea lo más seguro posible para los usuarios finales.

El código: Estándares de codificación

Otro aspecto importante es la estándar codificación. A fin de garantizar la calidad de Software y mantener la legibilidad de la fuente, la aplicación de estándares de codificación es una práctica común y necesaria. Así, en este proyecto, los principios de codificación limpia y legible se utilizan activamente en todo el sistema desarrollado con Python y Flask. Desde el primer inicio, la nomenclatura asignada a variables, funciones y rutas se mantuvo consistente y clara. Por lo tanto, se facilitó la comprensión del flujo lógico del programa por parte del desarrollador actual y futuro. La organización modulable también se puso en práctica.

Las distintas funciones tenidas en cuenta, la función de usuario actual en un token temporal, agregar registro e inicio de sesión y la gestión de productos se dividieron claramente. En consecuencia, no hubo duplicidad y se promovió la reutilización del código, como se muestra mediante la reutilización del decorador @token_required para proteger varias rutas sin copiar la lógica. Las funciones. A lo largo de todo este proceso, la indentación estándar de Python de cuatro tabulaciones es constante. Autoayuda para evitar errores de sintaxis y aumenta la legibilidad del código. Las rutas también se organizaron de forma clara y sencilla.

Las rutas RESTful simples se utilizan en todo el espectro, y los puntos finales se asignan a acciones explícitas. Ejemplo, /producto/agregar, /producto/editar/<id>, /producto/borrar/<id>, etc. De este modo, ayuda a los desarrolladores a comprender rápidamente la función del sistema y alinearse con otro software.

Además, se evitaron los literales de cadena innecesarios en las consultas SQL, utilizando parámetros posicionales (?) para mantener la seguridad y claridad en el código. Del mismo modo, la base de datos se inicializó de manera robusta con una función start_db(), que asegura que las tablas esenciales estén disponibles al iniciar la aplicación. De igual forma, se evitó que las inserciones iniciales se duplicaran mediante el uso de cláusulas WHERE NOT EXISTS.

En general, el código presenta una aplicación disciplinada de la separación de responsabilidades, el reúso de funciones y la aplicación de estructuras condicionales y ciclos. Como resultado no solo se observa mejor el rendimiento, sino que también se facilita la localización de errores, el mantenimiento y la escalabilidad futura del sistema.

Capturas

Registro

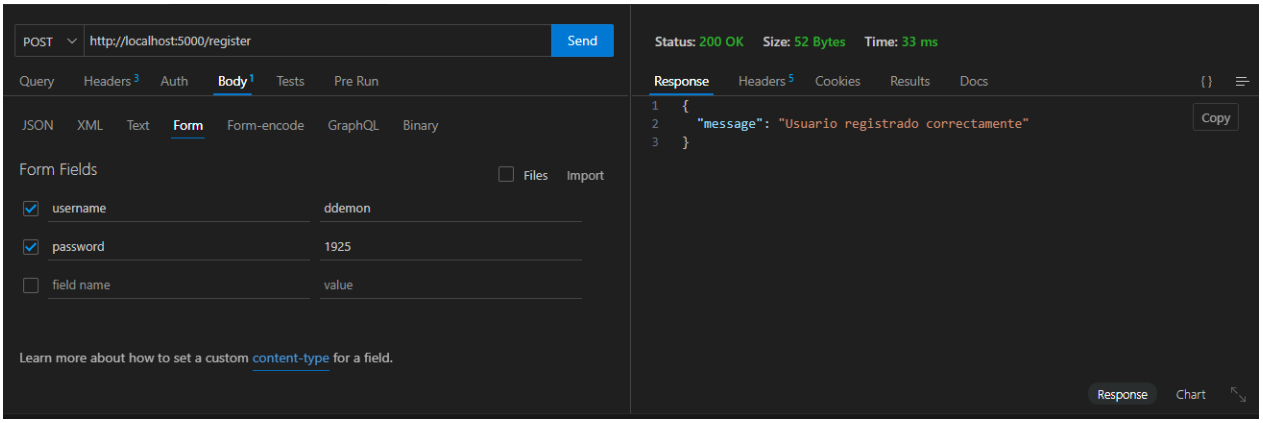

Figure 1: Registro

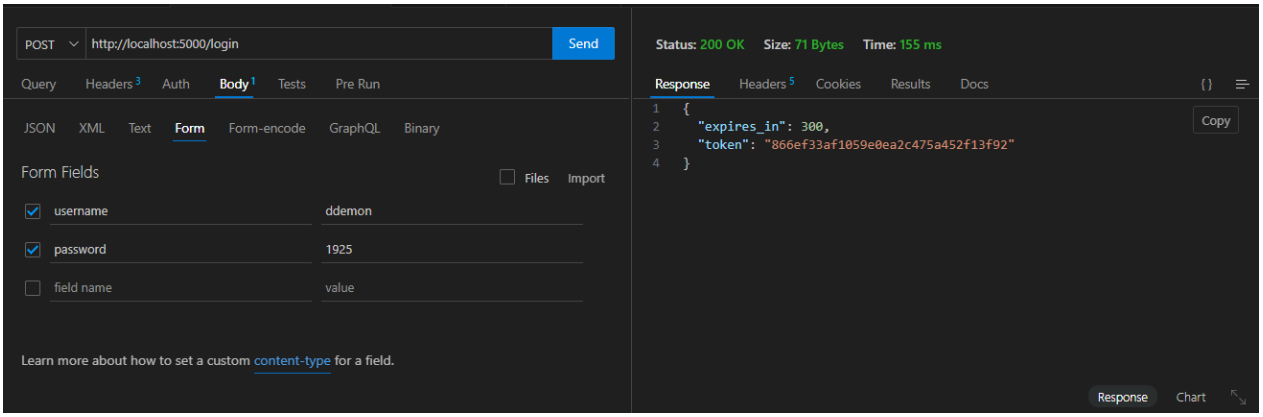

Figure 2: Login

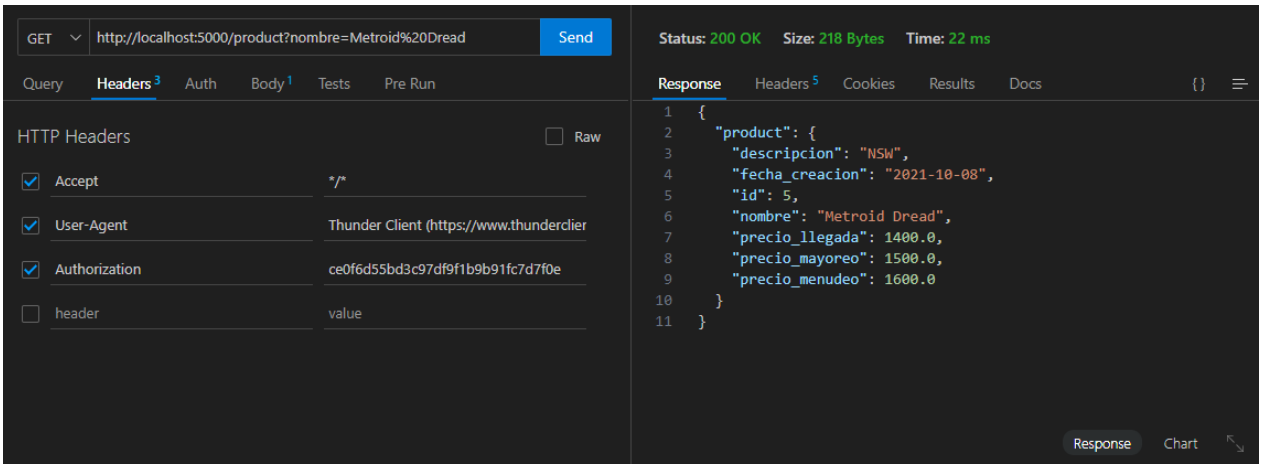

Figure 3: Mostrar Producto

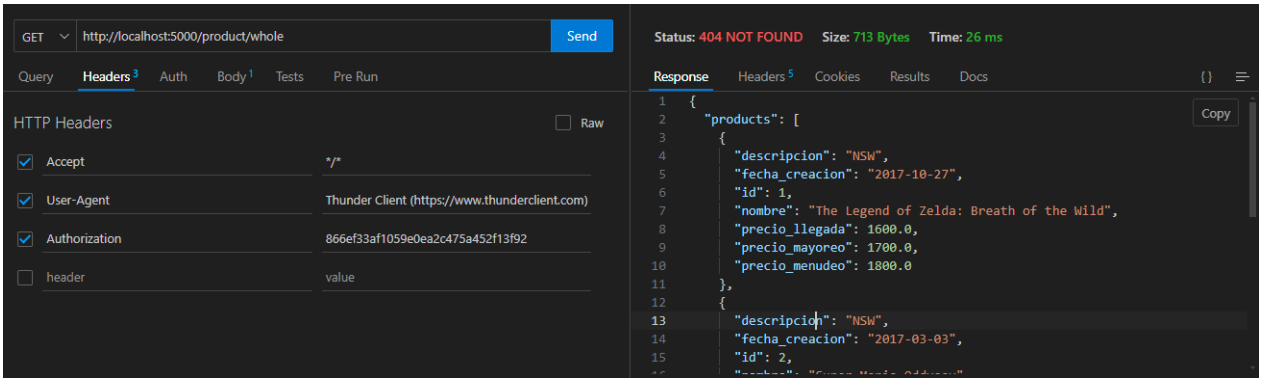

Figure 4: Mostrar Todos los Productos

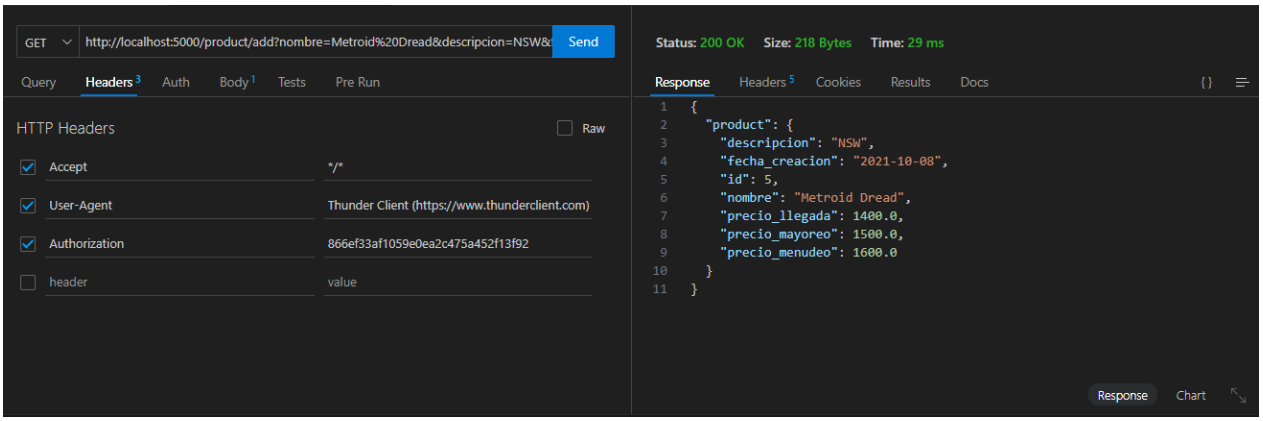

Figure 5: Añadir

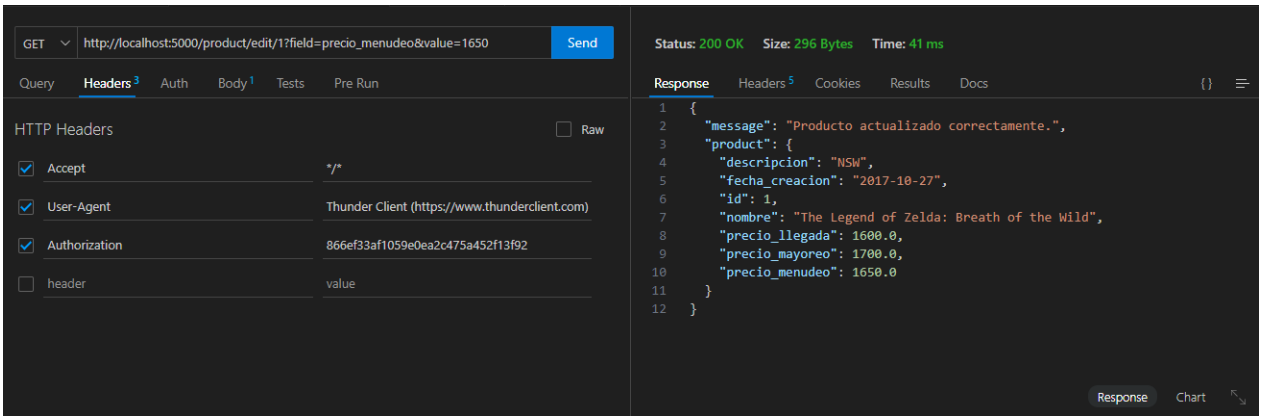

Figure 6: Editar

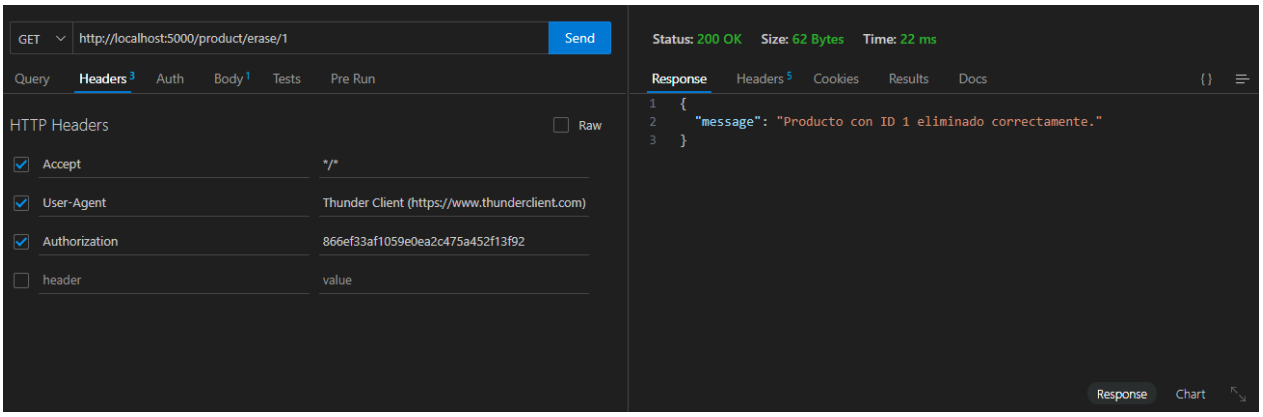

Figure 7: Borrar

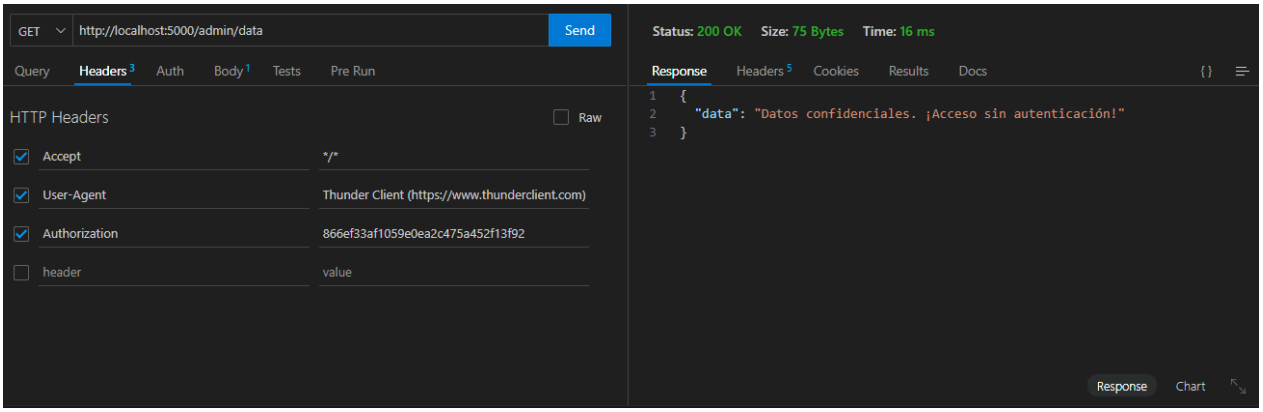

Figure 8: Datos Incorrectos/Faltantes

Conclusión

Como resultado final, en este proyecto, se desarrolla y se interactúa con una API RESTful utilizando Flask como framework del lado del servidor. La API expone una serie de rutas que permiten a los clientes consultar los productos almacenados en una base de datos y los devuelve en estructuras JSON. Finalmente, se implementó una capa de autenticación de pedido de tokens de una API, así como también en la aplicación frontend. Como se discutió anteriormente, en las aplicaciones actuales, es fundamental asegurar los datos y proteger no solo la integridad sino también los datos personales de los usuarios.

Durante las pruebas, que se implementaron mediante Thunder Client, hubo varios problemas comunes pero que en realidad solo confunden a los desarrolladores. Recibiendo un 404 estado por no encontrado, sin embargo, el contenido parece ser razonable. Por lo general, el servidor no devuelve ningún estado o, en palabras cortas, 200, o el nombre de la ruta ingresado en el cliente no coincide con el declarado en la API. Es decir, el servidor no puede encontrar correctamente la ruta solicitada. En esta implementación, faltaba el status code 200 en la función de transferencia de datos de grupos para incidencias en Flask.

Por otra parte, uno de los aprendizajes más importantes fue comprender que una respuesta con datos no implica que la solicitud fue exitosa desde el punto de vista del protocolo HTTP. El código de estado es un componente vital para la comunicación entre un cliente y un servidor, pues puede ser tanto si la solicitud fue exitosa, si hubo problemas con la autentificación o si el recurso solicitado no existe. Por esta razón, la correcta utilización de estos bajo el esquema usual de Flask return jsonify(...), 200 es vital para evitar malinterpretaciones desde el lado del cliente.

Finalmente, se recalca la importancia de utilizar herramientas de pruebas como Thunder Client para depurar el comportamiento de las APIs gratis entregadas antes de integrarlas a una aplicación más grande. Thunder Client permite simular peticiones HTTP con distintos métodos, headers personalizados como tokens JWT y desplegar de manera fácil las respuestas, haciendo que el proceso de corrección de errores sea más sencillo y eficiente.

En conclusión, este ejercicio ha permitido aprender por medio de la práctica conocimientos técnicos básicos sobre desarrollo web, rutas y autenticación en Flask, así como sobre las buenas prácticas en la creación y el tests en APIs. En mi experiencia, arreglar los errores mostrados refuerza la necesidad de mantener una arquitectura coherente, probar todos los flujos imaginables, y verificar que tanto el contenido como el estado respondan consistentemente con la lógica de la aplicación empresarial mantenida en el backend.

Bibliografias

- DigitalOcean. (2020). Cómo añadir autenticación a su aplicación con Flask-Login. Recuperado de https://www.digitalocean.com/community/tutorials/how-to-add-authentication-to-your-app-with-flask-login-es
- LabEx. (2023). Configuración de la base de datos SQLite con Python Flask. Recuperado de https://labex.io/es/tutorials/flask-python-flask-sqlite-database-setup-136336
- Medium.com – Diego Coder. (2024). Crea un sistema de autenticación en Flask con JSON Web Tokens. Recuperado de https://medium.com/@diego.coder/autenticación-en-python-con-json-web-tokens-y-flask-9f486bc88782
- 4Geeks Academy. (2025). Comprendiendo JWT y como implementar un JWT simple con Flask. Recuperado de https://4geeks.com/es/lesson/que-es-jwt-y-como-implementarlo-con-flask
- 4Geeks Academy. (2023). Autenticación basada en tokens de acceso para tu API. Recuperado de https://4geeks.com/es/lesson/ques-es-token-de-acceso-para-api
- 4Geeks Academy. (2021). Autenticación JWT con Flask y React.js. Recuperado de https://4geeks.com/es/interactive-coding-tutorial/jwt-authentication-with-flask-react-es
- DigitalOcean. (2019). Cómo crear una aplicación web usando Flask en Python 3. Recuperado de https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3-es
- Pallets Projects. (2025). Using SQLite 3 with Flask. Recuperado de https://flask.palletsprojects.com/en/stable/patterns/sqlite3/
- Snyk. (2024). How to secure Python Flask applications. Recuperado de https://snyk.io/blog/secure-python-flask-applications/
- CodingNomads. (2023). Flask Authentication for APIs: HTTP & Token Authentication. Recuperado de https://codingnomads.com/python-flask-authentication-http-token-authentication
- GeeksforGeeks. (2025). How to Build a Web App using Flask and SQLite in Python. Recuperado de https://www.geeksforgeeks.org/how-to-build-a-web-app-using-flask-and-sqlite-in-python/
- Curity. (2018). Securing a Python Flask API with JWTs. Recuperado de https://curity.io/resources/learn/oauth-filter-for-python-flask/

- YouTube – NelsonCode. (2023, marzo). JSON Web Token, REST API con Flask. Recuperado de https://www.youtube.com/watch?v=i1IwZ2X82nc
- YouTube – Rojas-Andres. (2021, mayo). Python - Flask - Añadiendo seguridad a nuestra API. Recuperado de https://www.youtube.com/watch?v=NvIZm-MvKIY
- YouTube – Tutorial Flask SQLAlchemy. (2018). Tutorial Flask – Lección 5: Base de datos con Flask SQLAlchemy. Recuperado de https://j2logo.com/tutorial-flask-leccion-5-base-de-datos-con-flask-sqlalchemy/