

Unidad III

***"Interfaces de Programación de aplicaciones en la
automatización de redes"***

**Documento para evaluar el Saber Hacer de
la materia de Automatización de
Infraestructura Digital I**

**Ingeniería
En
Redes Inteligentes y Ciberseguridad**

**Elaborado por:
Hannia Damaris Ramos Peñafort
Noemi Sanchez Pérez
Pamela Sherlin Silva Vargas**

**Maestro:
Gabriel Alejandro Reyes Morales**

Ramos Arizpe, Coahuila

24 de noviembre de 2024

Índice

Introducción	3
Evidencia gráfica de Postman utilizando las operaciones de agregar, buscar, actualizar y eliminar.	11
Tabla del diagnóstico de errores basado en la interpretación de códigos de estado.	
13	
Conclusión	16

Introducción

En este documento se han implementado operaciones en PYThon utilizando scripts en texto plano. El objetivo de este proyecto es crear una pequeña herramienta que permita al usuario realizar estas operaciones en una estructura de datos sencilla, como una lista o un diccionario este script permite agregar nuevos elementos a la estructura.

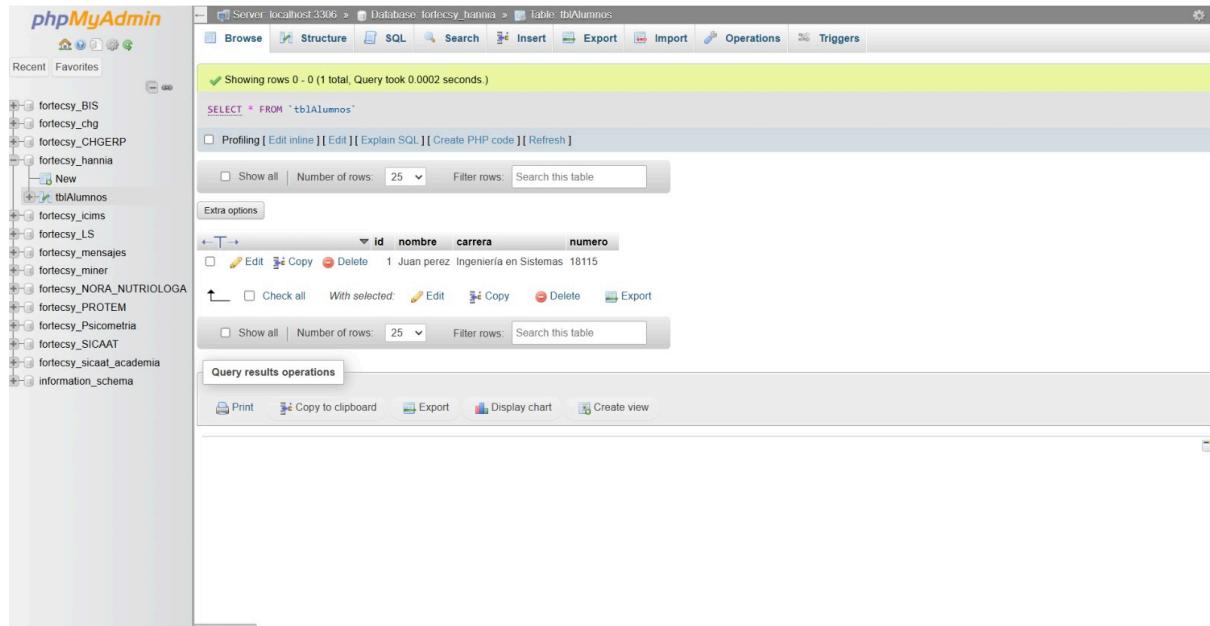
Buscar elementos por algún criterio determinado, actualizar los datos existentes con nueva información, eliminar elementos según el criterio de la búsqueda estas operaciones se manejan de manera eficiente y sencilla utilizando funciones de Python que proporcionan la base para aplicaciones más complejas.

Para probar la funcionalidad de las operaciones CRUD implementadas, se utilizó Postman, una herramienta ampliamente reconocida para el diseño, prueba y documentación de APIs. A través de esta plataforma, se realizaron solicitudes HTTP (como POST, GET, PUT y DELETE), confirmando que las operaciones de agregar, buscar, actualizar y eliminar funcionan como se espera. Las evidencias gráficas generadas muestran los datos enviados, las respuestas obtenidas y los códigos de estado correspondientes.

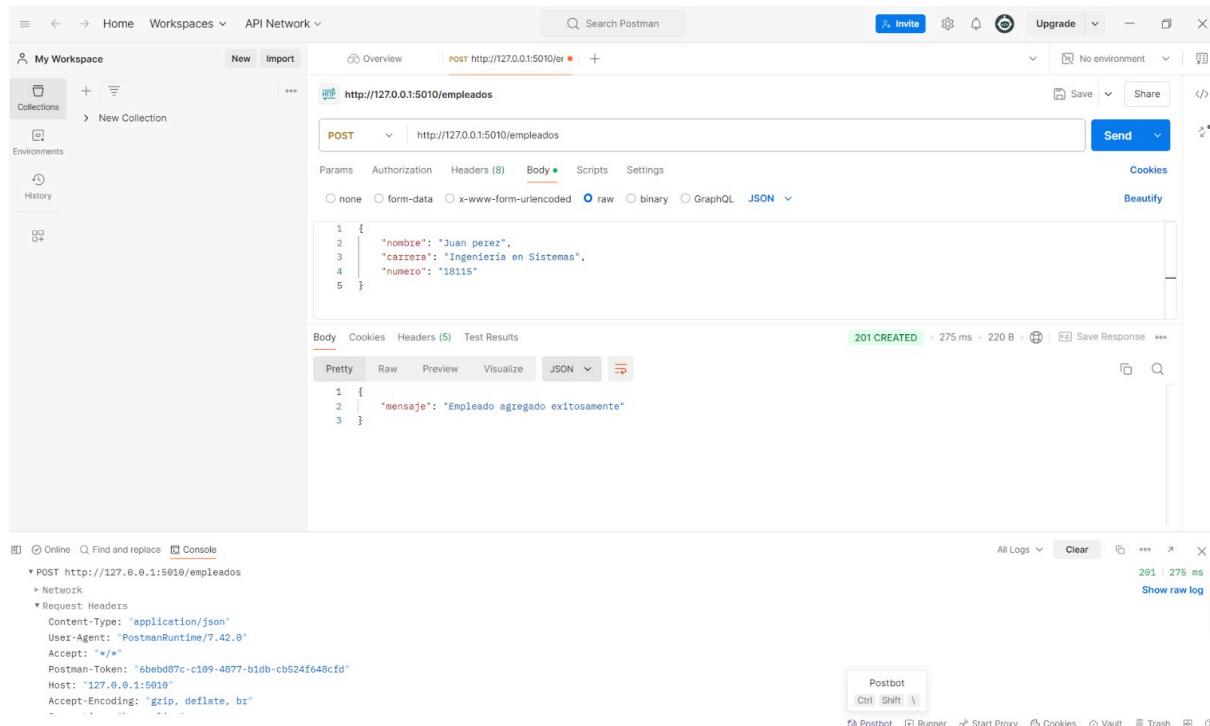
Se elaboró una tabla detallada que interpreta los códigos de estado HTTP obtenidos durante las pruebas realizadas en Postman. Esta tabla proporciona una referencia clara sobre el significado de cada código, incluyendo los casos de éxito (como el 200 OK) y los posibles errores (como el 404 Not Found o el 500 Internal Server Error). Además, identifica posibles causas de errores y ofrece recomendaciones para resolverlos, garantizando la depuración efectiva y la mejora continua del sistema.

Scripts en texto plano en el lenguaje Python utilizando las operaciones de agregar, buscar, actualizar y eliminar.

- Tabla en la base de datos



- Pantallas de la ejecución del código desde postman



POST http://127.0.0.1:5010/empleados

```
1 {  
2   "nombre": "Juan perez",  
3   "carrera": "Ingeniería en Sistemas",  
4   "numero": "18115"  
5 }
```

Body Cookies Headers (5) Test Results

```
1 {  
2   "mensaje": "Empleado agregado exitosamente"  
3 }
```

201 CREATED 275 ms 220 B Save Response

Request Headers

```
Content-Type: application/json  
User-Agent: PostmanRuntime/7.42.0  
Accept: */*  
Postman-Token: 6bebdb07c-c109-4877-b1db-cb524f648cfcd  
Host: 127.0.0.1:5010  
Accept-Encoding: gzip, deflate, br
```

phpMyAdmin

Server: localhost:3306 » Database: fortency_hannia » Table: tbIImpleados

Browse Structure SQL Search Insert Export Import Operations Triggers

Recent Favorites

Showing rows 0 - 1 (total, Query took 0.0003 seconds.)

SELECT * FROM `tbIImpleados`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	id	nombre	carrera	numero
<input type="checkbox"/>	1	Juan perez	Ingeniería en Sistemas	18115

Edit Copy Delete Check all With selected Edit Copy Delete Export

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for Home, Workspaces, and API Network. The main workspace is titled "My Workspace". A search bar at the top right contains the placeholder "Search Postman". On the left sidebar, there are sections for Collections, Environments, and History. The main content area displays a collection named "New Collection". A request card for a GET method to "http://127.0.0.1:5010/empleados" is shown. The "Body" tab is selected, displaying the response as JSON. The response body is:

```
200 OK
[{"id": 1, "nombre": "Juan perez", "numero": "18115", "carrera": "Ingenieria en Sistemas"}, {"id": 3, "nombre": "Hannia Ramos penafor", "numero": "18111", "carrera": "Ingenieria en Sistemas"}]
```

The bottom section of the interface includes a network tab with a single entry for the GET request, and a status bar showing "200 | 285 ms" and a "Show raw log" link.

- Verificación de la inserción de datos

Server: localhost:3306 > Database: fortecsy_hannia > Table: tblEmpleados

Browse Structure SQL Search Insert Export Import Operations Triggers

Showing rows 0 - 1 (2 total, Query took 0.0004 seconds.)

SELECT * FROM `tblEmpleados`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	id	nombre	carrera	numero
<input type="checkbox"/>	1	Juan perez	Ingeniería en Sistemas	18115
<input type="checkbox"/>	3	Hannia Ramos penafort	Ingeniería en Sistemas	18111

Check all With selected: Edit Copy Delete Export

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

Home Workspaces API Network

PUT http://127.0.0.1:5010/empleados/1

Params Authorization Headers (8) Body Scripts Settings

```

1 {
2   "nombre": "Agustin zamora",
3   "carrera": "Ingenieria en Sistemas",
4   "numero": "1221"
5 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "empleado": {
3     "carrera": "Ingenieria en Sistemas",
4     "id": 1,
5     "nombre": "Agustin zamora",
6     "numero": "1221"
7   },
8   "mensaje": "Empleado actualizado exitosamente"
9 }
```

200 OK · 643 ms · 352 B · Save Response

Online Find and replace Console

PUT http://127.0.0.1:5010/empleados/1

Request Headers

- Content-Type: application/json
- User-Agent: PostmanRuntime/7.42.0
- Accept: */*
- Postman-Token: b74ba070-0ea1-4745-9094-bb4ed1ba27f9
- Host: 127.0.0.1:5010
- Accept-Encoding: gzip, deflate, br

All Logs Clear

200 643 ms Show raw log

Postbot Ctrl Shift ↵

Server: localhost:3306 Database: forcecsy_hannia Table: tblEmpleados

Browse Structure SQL Search Insert Export Import Operations Triggers

Showing rows 0 - 1 (2 total, Query took 0 0003 seconds.)

SELECT * FROM `tblEmpleados`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input checked="" type="checkbox"/> id	<input checked="" type="checkbox"/> nombre	<input checked="" type="checkbox"/> carrera	<input checked="" type="checkbox"/> numero
1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	Agustin zamora	Ingenieria en Sistemas	1221
2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	3	Hannia Ramos penafor	Ingenieria en Sistemas	18111

Check all With selected: Edit Copy Delete Export

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

My Workspace

Overview

http://127.0.0.1:5010/empleados/1

DELETE http://127.0.0.1:5010/empleados/1

Params Authorization Headers (8) Body Scripts Settings

Body (JSON)

```

1 {
2   "nombre": "Agustin Zamora",
3   "carrera": "Ingeniería en Sistemas",
4   "numero": "1221"
5 }

```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "mensaje": "Empleado eliminado exitosamente"
3 }

```

200 OK 382 ms 216 B Save Response

Request Headers

Network

Request Headers

Content-Type: application/json
User-Agent: PostmanRuntime/7.42.0
Accept: */*
Postman-Token: b8388eaa-241f-4ebd-8e2a-132d59db5b78
Host: 127.0.0.1:5010
Accept-Encoding: gzip, deflate, br

All Logs Clear 200 382 ms Show raw log

Browse Structure SQL Search Insert Export Import Operations Triggers

Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

SELECT * FROM `tblEmpleados`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table

Extra options

	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	With selected:	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input type="checkbox"/> Export
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	3 Hannia Ramos penafont	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Show all Number of rows: 25 Filter rows: Search this table

Query results operations

Print Copy to clipboard Export Display chart Create view

Código

```

from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

# Configuración de la conexión a MySQL/MariaDB
app.config['SQLALCHEMY_DATABASE_URI'] =
    'mysql+pymysql://{}USER:{}PASSWORD@{}SERVER}/{DATABASE}'.
).format(

```

```

USER='fortecsy_practicas',
PASSWORD='h4nn1aR4m05',
SERVER='www.fortecsystems.com.mx', # Cambia según tu configuración
DATABASE='fortecsy_hannia'

)
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

# Modelo de la tabla
class Empleado(db.Model):
    _tablename_ = 'tblEmpleados' # Cambia por el nombre de tu tabla

    id = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(100), nullable=False)
    carrera = db.Column(db.String(100), nullable=False)
    numero = db.Column(db.String(10), unique=True, nullable=False)

    def to_dict(self):
        return {
            'id': self.id,
            'nombre': self.nombre,
            'carrera': self.carrera,
            'numero': self.numero
        }

# Ruta para obtener todos los empleados
@app.route('/empleados', methods=['GET'])
def obtener_empleados():
    try:
        empleados = Empleado.query.all()
        empleados_lista = [empleado.to_dict() for empleado in empleados]
        return jsonify(empleados_lista), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

```

```
# Ruta para agregar un nuevo empleado (POST)
@app.route('/empleados', methods=['POST'])
def agregar_empleado():
    data = request.json # Obtener datos del cuerpo de la solicitud
    try:
        nuevo_empleado = Empleado(
            nombre=data['nombre'],
            carrera=data['carrera'],
            numero=data['numero']
        )
        db.session.add(nuevo_empleado)
        db.session.commit()
        return jsonify({"mensaje": "Empleado agregado exitosamente"}), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 400

# Ruta para actualizar un empleado por ID (PUT)
@app.route('/empleados/<int:id>', methods=['PUT'])
def actualizar_empleado(id):
    data = request.json # Obtener datos del cuerpo de la solicitud
    try:
        empleado = Empleado.query.get(id)
        if not empleado:
            return jsonify({"error": "Empleado no encontrado"}), 404

        # Actualizar campos
        empleado.nombre = data.get('nombre', empleado.nombre)
        empleado.carrera = data.get('carrera', empleado.carrera)
        empleado.numero = data.get('numero', empleado.numero)

        db.session.commit()
    
```

```

        return jsonify({"mensaje": "Empleado actualizado exitosamente", "empleado": empleado.to_dict()}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 400

# Ruta para eliminar un empleado por ID (DELETE)
@app.route('/empleados/<int:id>', methods=['DELETE'])
def eliminar_empleado(id):
    try:
        empleado = Empleado.query.get(id)
        if not empleado:
            return jsonify({"error": "Empleado no encontrado"}), 404

        db.session.delete(empleado)
        db.session.commit()
        return jsonify({"mensaje": "Empleado eliminado exitosamente"}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 400

if __name__ == '__main__':
    app.run(debug=True, port=5010)

```

Datos de base de datos

```

CREATE TABLE tblAlumnos (
    id INT PRIMARY KEY IDENTITY(1,1),
    nombre NVARCHAR(100) NOT NULL,
    carrera NVARCHAR(100) NOT NULL,
    numero NVARCHAR(10) NOT NULL
);

```

Evidencia gráfica de Postman utilizando las operaciones de agregar, buscar, actualizar y eliminar.

The screenshot shows the Postman interface with a successful POST request to `http://127.0.0.1:5010/alumnos`. The request body contains:

```
1 {
2   "nombre": "Hannia Damaris Ramos Penafont",
3   "carrera": "Ingenieria en Sistemas",
4   "numero": "18111"
5 }
```

The response status is `201 CREATED` with a duration of `142 ms` and a size of `218 B`. The response body is:

```
1 {
2   "mensaje": "Alumno agregado exitosamente"
3 }
```

The request details in the logs show the JSON payload sent to the server.

The screenshot shows the Postman interface with a successful GET request to `http://127.0.0.1:5010/alumnos`. The response status is `200 OK` with a duration of `91 ms` and a size of `427 B`. The response body is a list of students:

```
1 [
2   {
3     "carrera": "Ingenieria en Sistemas",
4     "id": 1,
5     "nombre": "Hannia Damaris Ramos Penafont",
6     "numero": "18111"
7   },
8   {
9     "carrera": "Ingenieria en Sistemas",
10    "id": 2,
11    "nombre": "JUANITO PEREZ",
12    "numero": "18112"
13 }
```

The request details in the logs show the JSON payload sent to the server.

My Workspace

PUT http://127.0.0.1:5010/alumnos/2

Body (8) Body

```
1 {
2   "nombre": "Juan perez",
3   "carreira": "Ingenieria en Sistemas",
4   "numero": "18115"
5 }
```

200 OK 440 ms 345 B Save Response

PUT http://127.0.0.1:5010/alumnos/2

Request Headers

```
Content-Type: application/json
User-Agent: PostmanRuntime/7.42.0
Accept: */*
Postman-Token: 171ee934-c4f7-44d8-8323-ba32300d19aa
Host: 127.0.0.1:5010
Accept-Encoding: gzip, deflate, br
```

All Logs Clear 200 | 440 ms Show raw log

My Workspace

DELETE http://127.0.0.1:5010/alumnos/2

Body (8) Body

```
1 {
2   "nombre": "Juan perez",
3   "carreira": "Ingenieria en Sistemas",
4   "numero": "18115"
5 }
```

200 OK 232 ms 214 B Save Response

DELETE http://127.0.0.1:5010/alumnos/2

Request Headers

```
Content-Type: application/json
User-Agent: PostmanRuntime/7.42.0
Accept: */*
Postman-Token: 68867460-ea38-4c29-adc7-7e666c773ed3
Host: 127.0.0.1:5010
Accept-Encoding: gzip, deflate, br
```

All Logs Clear 200 | 232 ms Show raw log

Tabla del diagnóstico de errores basado en la interpretación de códigos de estado.

Código de Estado	Descripción	Diagnóstico / Solución
500 Internal Server Error	Error de importación de librerías (<code>ImportError</code>).	Diagnóstico: Librerías necesarias como <code>flask_sqlalchemy</code> o <code>pymysql</code> no están instaladas. Solución: Instalar dependencias con <code>pip install flask_sqlalchemy pymysql</code> .
500 Internal Server Error	Uso incorrecto de <code>_name_</code> en lugar de <code>__name__</code> .	Diagnóstico: <code>_name_</code> está mal definido en <code>app = Flask(_name_)</code> . Solución: Reemplazar <code>_name_</code> por <code>__name__</code> en el constructor de Flask.
500 Internal Server Error	Configuración incorrecta de <code>SQLALCHEMY_DATABASE_URI</code> .	Diagnóstico: Error en las credenciales (usuario, contraseña, servidor o base de datos). Solución: Verificar las credenciales y la conectividad con la base de datos.
404 Not Found	Ruta solicitada no definida en la aplicación.	Diagnóstico: El cliente intenta acceder a una ruta inexistente. Solución: Verificar que la URL solicitada coincida con una ruta configurada en la aplicación (<code>/empleados</code> , <code>/empleados/<int:id></code>).
400 Bad Request	Error al procesar datos JSON.	Diagnóstico: El cuerpo de la solicitud no incluye los campos esperados (<code>nombre</code> , <code>carrera</code> ,

		<p>numero).</p> <p>Solución: Validar los datos enviados por el cliente antes de procesarlos.</p>
404 Not Found	Empleado no encontrado en una operación de <code>PUT</code> o <code>DELETE</code> .	<p>Diagnóstico: El ID proporcionado no existe en la base de datos.</p> <p>Solución: Validar la existencia del empleado antes de intentar modificarlo o eliminarlo.</p>
400 Bad Request	Violación de la restricción de unicidad (<code>IntegrityError</code>) en la columna <code>numero</code> .	<p>Diagnóstico: Se intenta agregar o actualizar un registro con un número duplicado.</p> <p>Solución: Implementar validaciones para evitar duplicados antes de guardar los datos.</p>
500 Internal Server Error	Error en la declaración del modelo SQLAlchemy (<code>_tablename_</code> en lugar de <code>--tablename--</code>).	<p>Diagnóstico: SQLAlchemy no reconoce el atributo <code>_tablename_</code>.</p> <p>Solución: Cambiar <code>_tablename_</code> por <code>--tablename--</code> en la definición del modelo <code>Empleado</code>.</p>
500 Internal Server Error	Diferencias entre el modelo SQLAlchemy y la estructura de la tabla en la base de datos.	<p>Diagnóstico: Las columnas del modelo no coinciden con las definidas en la tabla de la base de datos (<code>tblEmpleados</code>).</p> <p>Solución: Asegurarse de que las columnas y sus tipos en la base de datos coincidan con las definiciones del modelo SQLAlchemy.</p>
500 Internal Server Error	Error de tipos al manejar valores (<code>OperationalError</code> o <code>DataError</code>).	Diagnóstico: Un campo tiene un tipo de dato diferente al esperado (por ejemplo, enviar texto a una columna tipo INT).

		Solución: Validar y, si es necesario, convertir los datos enviados antes de procesarlos.
500 Internal Server Error	Error al serializar un objeto no compatible con JSON.	Diagnóstico: El método <code>to_dict</code> devuelve tipos de datos no serializables por JSON. Solución: Asegurar que <code>to_dict</code> solo devuelva valores como <code>str</code> , <code>int</code> , o <code>float</code> , y evitar objetos complejos.

Conclusión

Este proyecto consolidó habilidades técnicas en programación, pruebas y diagnóstico, destacando la importancia de un flujo de trabajo bien estructurado para el desarrollo de sistemas basados en servicios web.

La combinación de Python, Postman y análisis de códigos de estado resalta cómo las herramientas y técnicas modernas trabajan en conjunto para asegurar la funcionalidad, confiabilidad y eficiencia en la gestión de datos.

En conclusión la integración de scripts en Python, la evidencia práctica mediante Postman y el análisis de códigos de estado representa un enfoque integral para desarrollar, probar y diagnosticar aplicaciones que manejan operaciones CRUD (Crear, Leer, Actualizar y Eliminar). Estos elementos reflejan un entendimiento profundo de cómo construir y evaluar APIs funcionales y confiables.

La implementación de operaciones CRUD en texto plano utilizando Python demuestra la capacidad de estructurar y manejar datos mediante código eficiente y comprensible. Este proceso ilustra cómo diseñar scripts para interactuar con bases de datos o sistemas de almacenamiento, con un enfoque en la flexibilidad y escalabilidad.

La interpretación de los códigos de estado (200, 404, 500, entre otros) en la tabla de diagnóstico proporciona una guía clara para identificar y solucionar errores en el sistema. Esto asegura una experiencia de usuario mejorada al reducir fallos en el funcionamiento de la API.