Universidad Tecnológica de Coahuila

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

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.

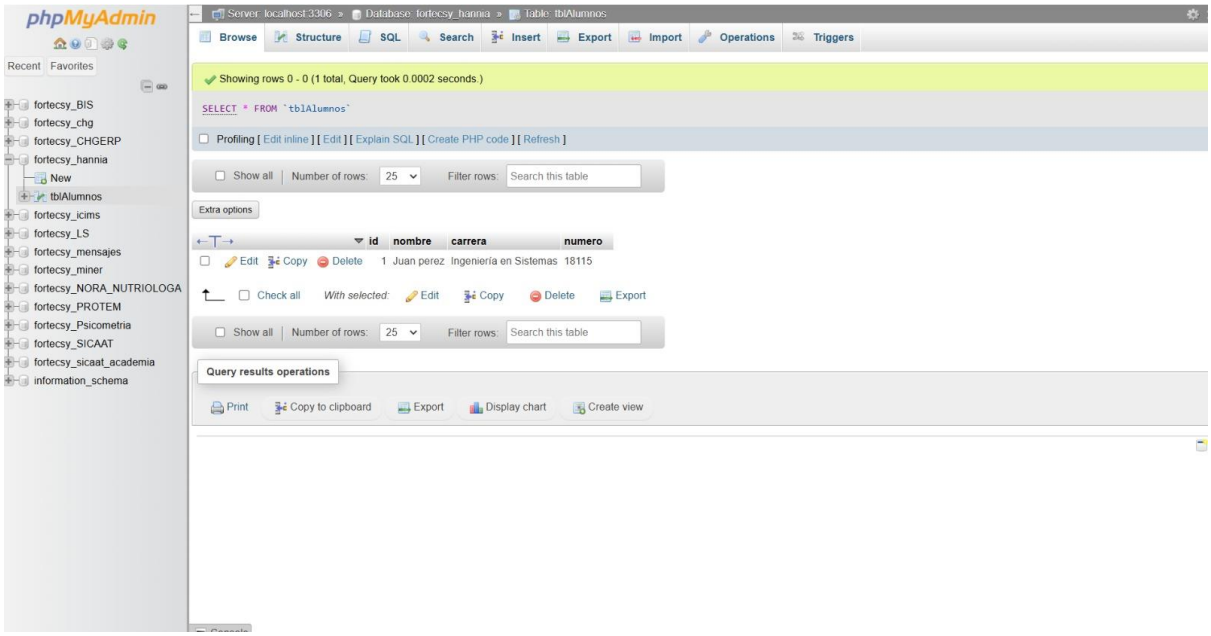
Buscar elementos por algún criterio determinado, actualizar los datos existentes con nueva información, eliminar elementos según el criterio de la búsqueda estas operaciones se manejan de manera eficiente y sencilla utilizando funciones de Python que proporcionan la base de para aplicaciones más complejas.

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**

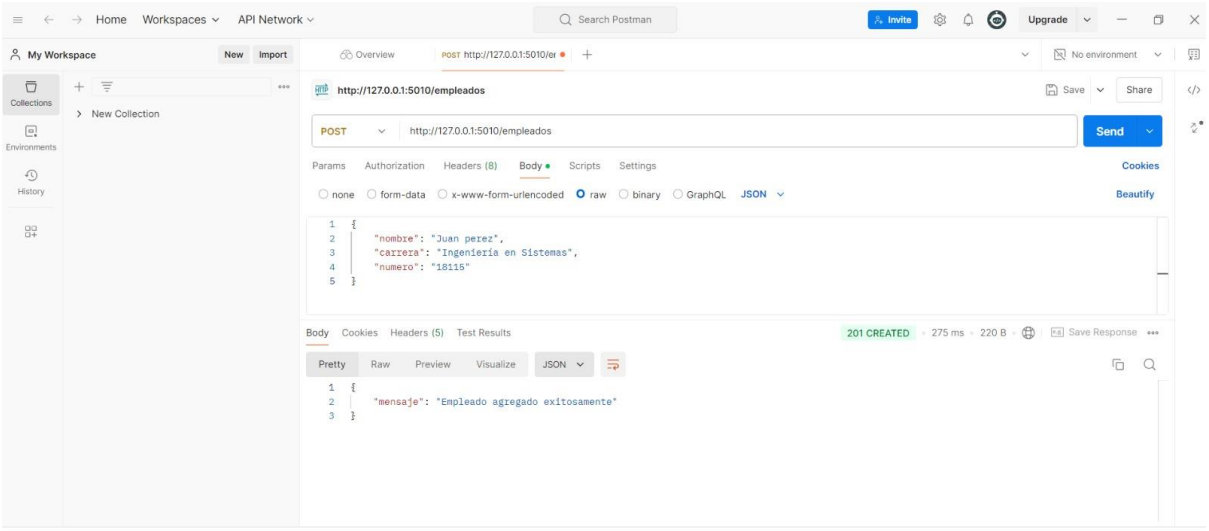
  http:127.0.0.1:5010/empleados

  -Guardamos el archivo con terminación **.py**

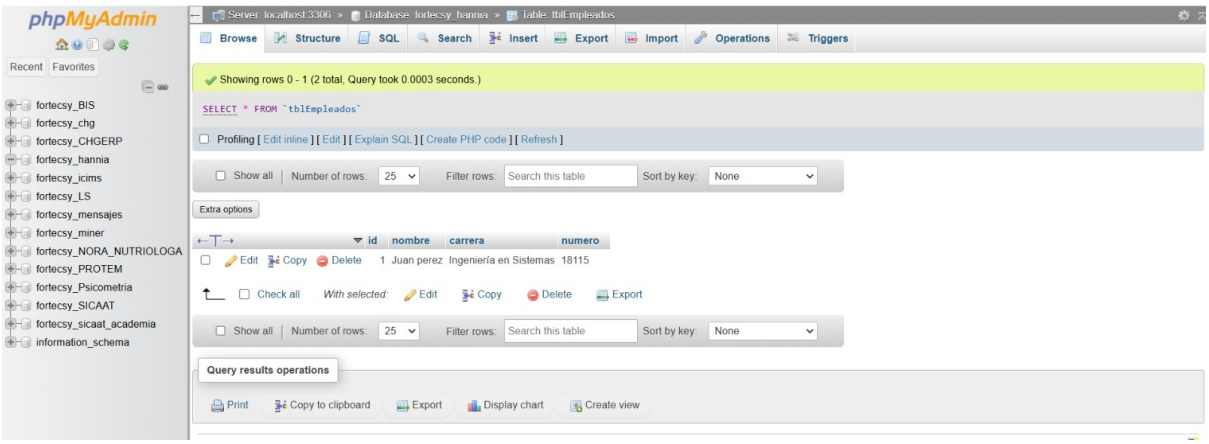
  -En la terminal, navegamos a la carpeta donde guardamos el archivo y ejecutamos el comando:

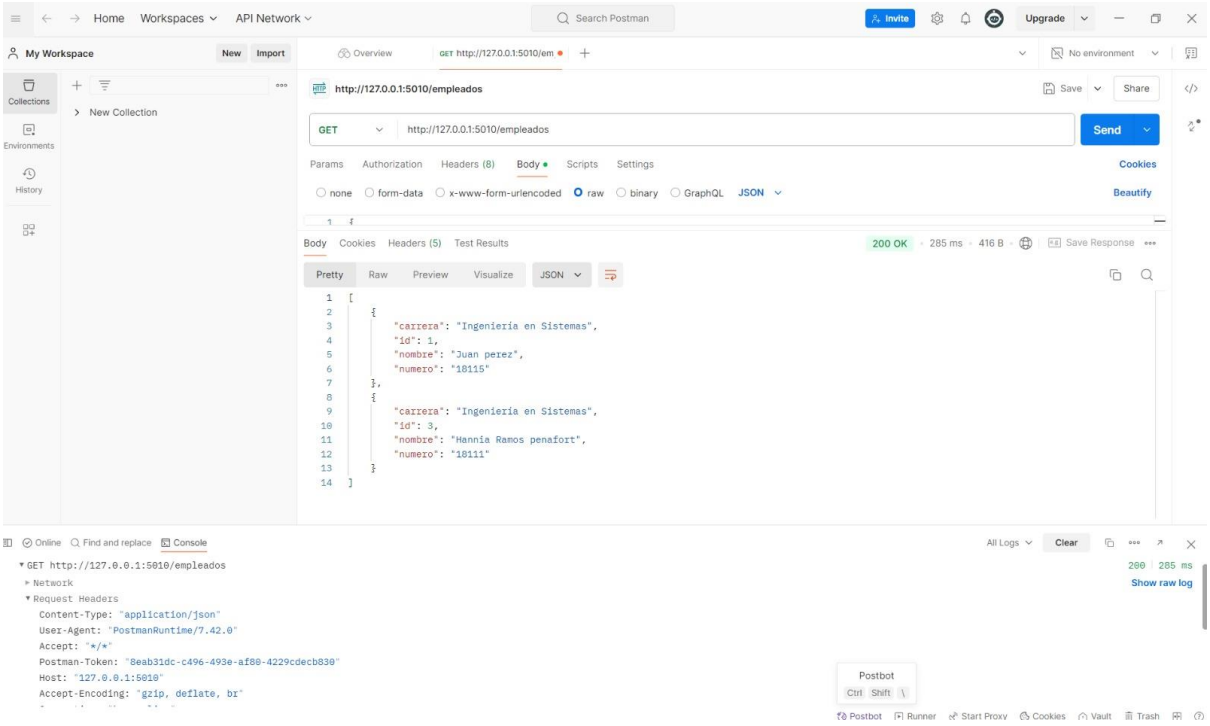
  python <nombre-del-archivo>.py

  Esto iniciará el servidor Flask en http://127.0.0.1:5010 o http://localhost:5010

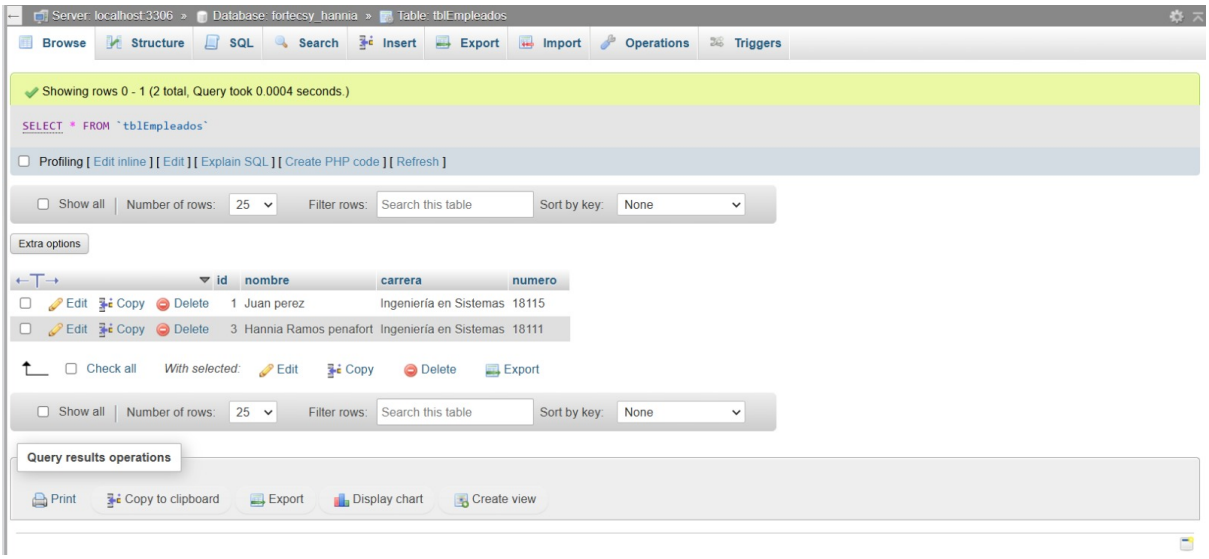

Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Environments
History
New Collection
http://127.0.0.1:5010/empleados
POST
http://127.0.0.1:5010/empleados
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nombre": "Juan perez",
"carrera": "Ingeniería en Sistemas",
"numero": "18115"
}
Body
Cookies
Headers (5)
Test Results
201 CREATED
275 ms
220 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"mensaje": "Empleado agregado exitosamente"
}

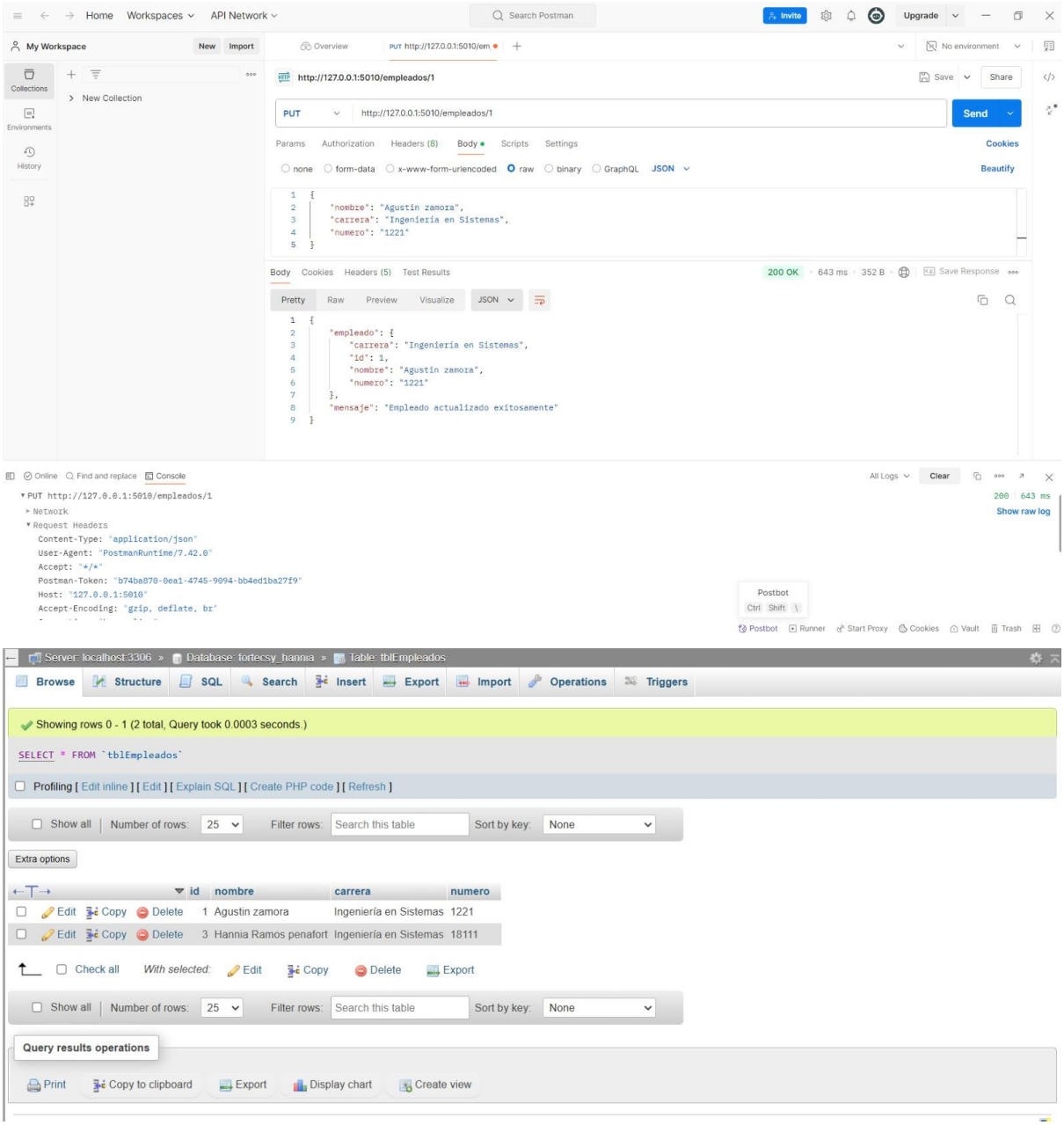

Online
Find and replace
Console
All Logs
Clear
POST http://127.0.0.1:5010/empleados
201
275 ms
Network
Show raw log
Request Headers
Content-Type: "application/json"
User-Agent: "PostmanRuntime/7.42.0"
Accept: "*/*"
Postman-Token: "6bebd87c-c109-4877-b1db-cb524f648cfd"
Host: "127.0.0.1:5010"
Accept-Encoding: "gzip, deflate, br"
Postbot
Ctrl Shift \
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


phpMyAdmin
Recent
Favorites
fortecsy_BIS
fortecsy_chg
fortecsy_CHGERP
fortecsy_hannia
fortecsy_icims
fortecsy_LS
fortecsy_mensajes
fortecsy_miner
fortecsy_NORA_NUTRIOLOGA
fortecsy_PROTEM
fortecsy_Psicometria
fortecsy_SICAAT
fortecsy_sicaat_academia
information_schema
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 0.0003 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
id
nombre
carrera
numero
Edit
Copy
Delete
1
Juan perez
Ingeniería en Sistemas
18115
Check all
With selected:
Edit
Copy
Delete
Export
Query results operations
Print
Copy to clipboard
Export
Display chart
Create view

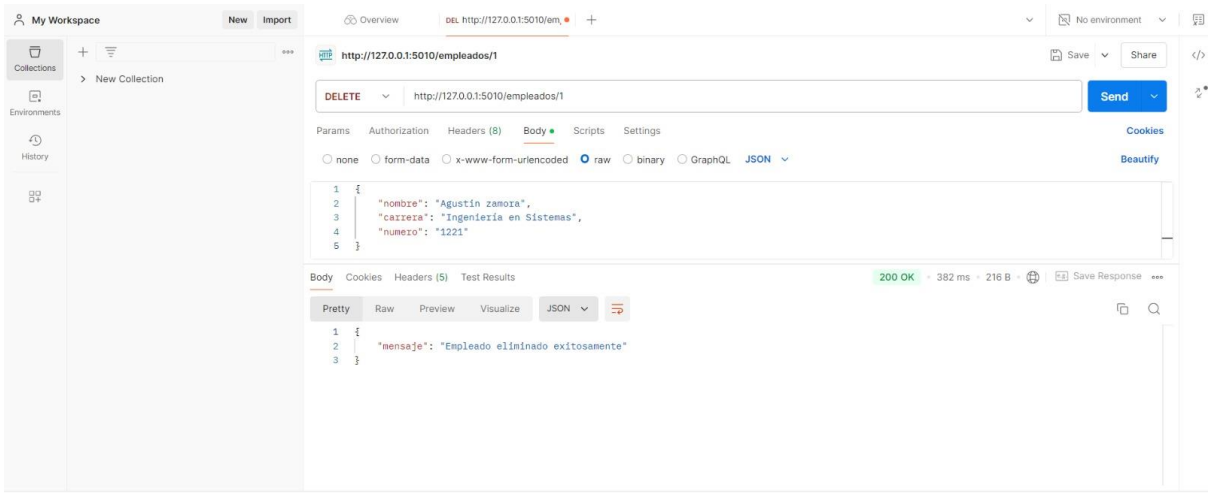


- Verificación de la inserción de datos

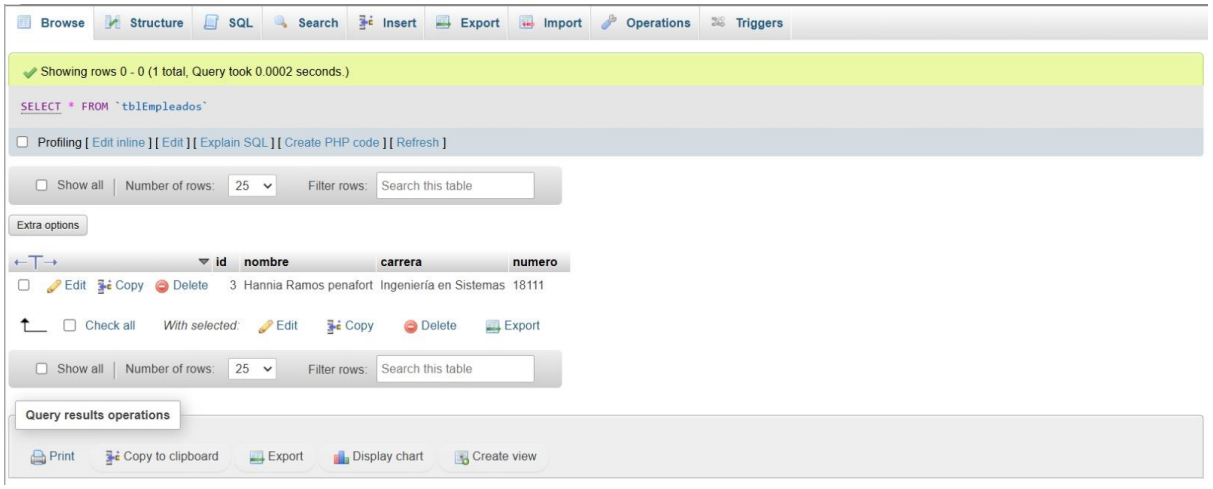



Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Environments
History
New Collection
Overview
PUT http://127.0.0.1:5010/em
No environment
http://127.0.0.1:5010/empleados/1
Save
Share
PUT
http://127.0.0.1:5010/empleados/1
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nombre": "Agustin zamora",
"carrera": "Ingeniería en Sistemas",
"numero": "1221"
}
Body
Cookies
Headers (5)
Test Results
200 OK
643 ms
352 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"empleado": {
"carrera": "Ingeniería en Sistemas",
"id": 1,
"nombre": "Agustin zamora",
"numero": "1221"
},
"mensaje": "Empleado actualizado exitosamente"
}
Online
Find and replace
Console
All Logs
Clear
PUT http://127.0.0.1:5010/empleados/1
200
643 ms
Network
Show raw log
Request Headers
Content-Type: "application/json"
User-Agent: "PostmanRuntime/7.42.0"
Accept: "*/*"
Postman-Token: "b74ba870-0ea1-4745-9094-bb4ed1ba27f9"
Host: "127.0.0.1:5010"
Accept-Encoding: "gzip, deflate, br"
Postbot
Ctrl Shift \
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

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 0.0003 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
id
nombre
carrera
numero
Edit Copy Delete 1 Agustin zamora Ingeniería en Sistemas 1221
Edit Copy Delete 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


My Workspace
New
Import
Overview
http://127.0.0.1:5010/em
No environment
Collections
Environments
History
New Collection
http://127.0.0.1:5010/empleados/1
Save
Share
DELETE
http://127.0.0.1:5010/empleados/1
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nombre": "Agustín zamora",
"carrera": "Ingeniería en Sistemas",
"numero": "1221"
}
Body
Cookies
Headers (5)
Test Results
200 OK
382 ms
216 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"mensaje": "Empleado eliminado exitosamente"
}


Online
Find and replace
Console
All Logs
Clear
DELETE http://127.0.0.1:5010/empleados/1
200
382 ms
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"
Show raw log
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


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
id
nombre
carrera
numero
Edit
Copy
Delete
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
Query results operations
Print
Copy to clipboard
Export
Display chart
Create view

Codigo

```
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,
```

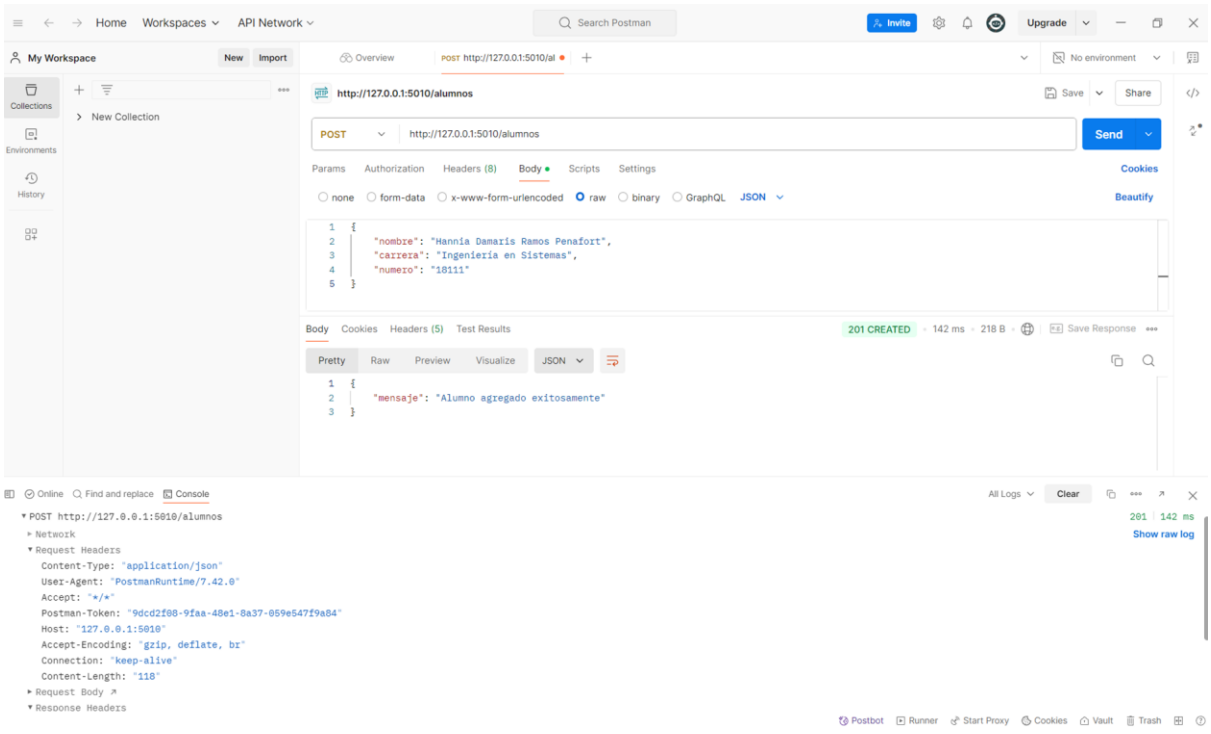
```
            'numero': self.numero
        }

# 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
```

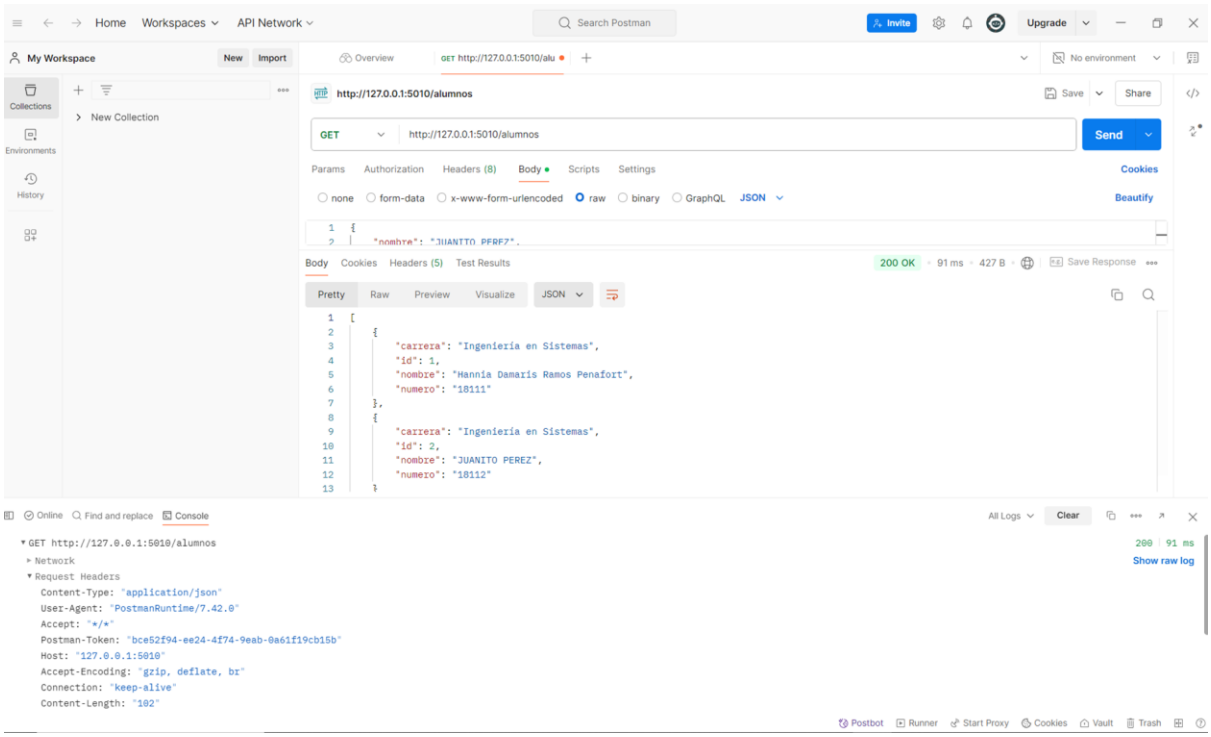
```
    try:
        empleado = Empleado.query.get(id)
        if not empleado:
            return jsonify({"error": "Empleado no encontrado"}), 404

        # 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)
```

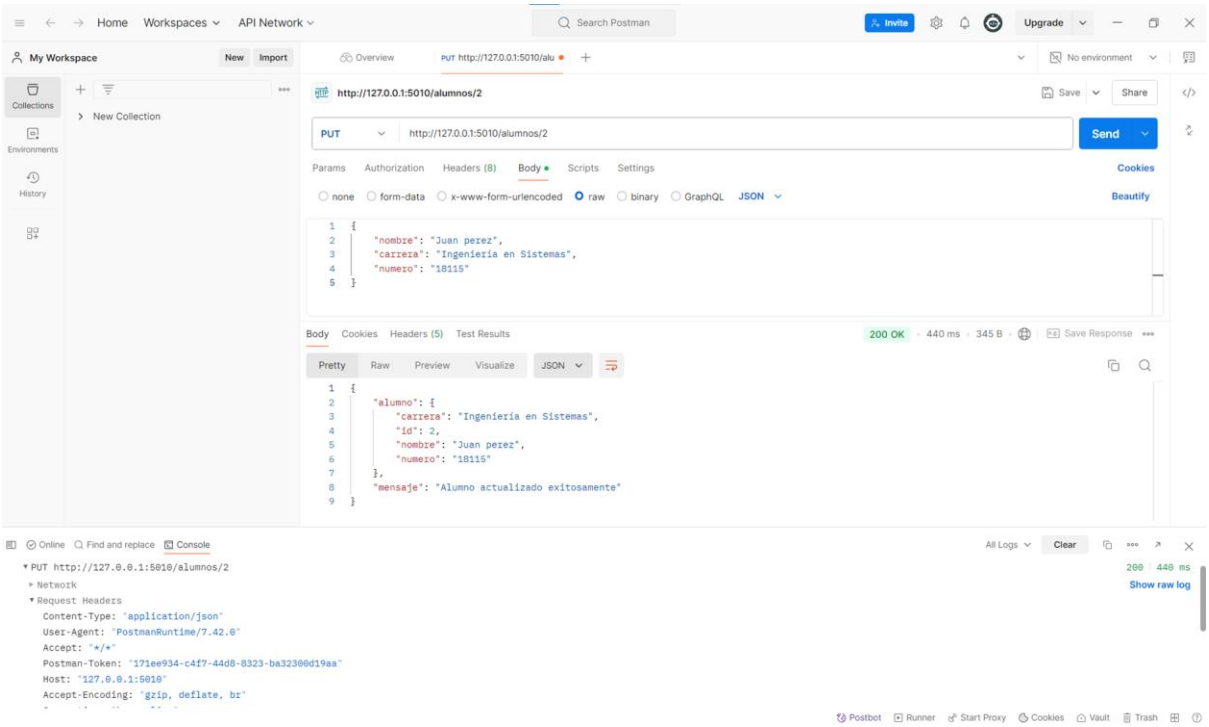
Datos de base de datos

```
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
);
```

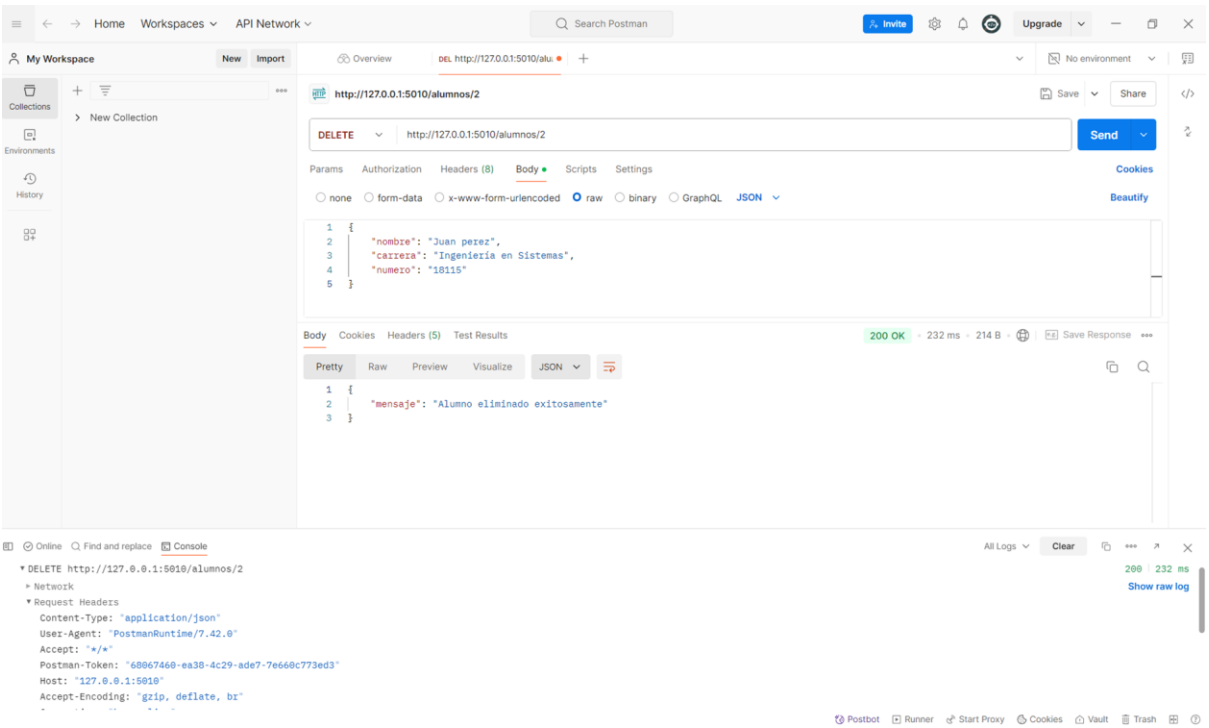
Evidencia gráfica de Postman utilizando las operaciones de agregar, buscar, actualizar y eliminar.






Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
PUT http://127.0.0.1:5010/alu
No environment
Collections
Environments
History
New Collection
http://127.0.0.1:5010/alumnos/2
Save
Share
PUT
http://127.0.0.1:5010/alumnos/2
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nombre": "Juan perez",
"carrera": "Ingeniería en Sistemas",
"numero": "18115"
}
Body
Cookies
Headers (5)
Test Results
200 OK
440 ms
345 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"alumno": {
"carrera": "Ingeniería en Sistemas",
"id": 2,
"nombre": "Juan perez",
"numero": "18115"
},
"mensaje": "Alumno actualizado exitosamente"
}

Online
Find and replace
Console
All Logs
Clear
PUT http://127.0.0.1:5010/alumnos/2
200
440 ms
Network
Show raw log
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"
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
DEL http://127.0.0.1:5010/alu
No environment
Collections
Environments
History
New Collection
http://127.0.0.1:5010/alumnos/2
Save
Share
DELETE
http://127.0.0.1:5010/alumnos/2
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nombre": "Juan perez",
"carrera": "Ingeniería en Sistemas",
"numero": "18115"
}
Body
Cookies
Headers (5)
Test Results
200 OK
232 ms
214 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"mensaje": "Alumno eliminado exitosamente"
}

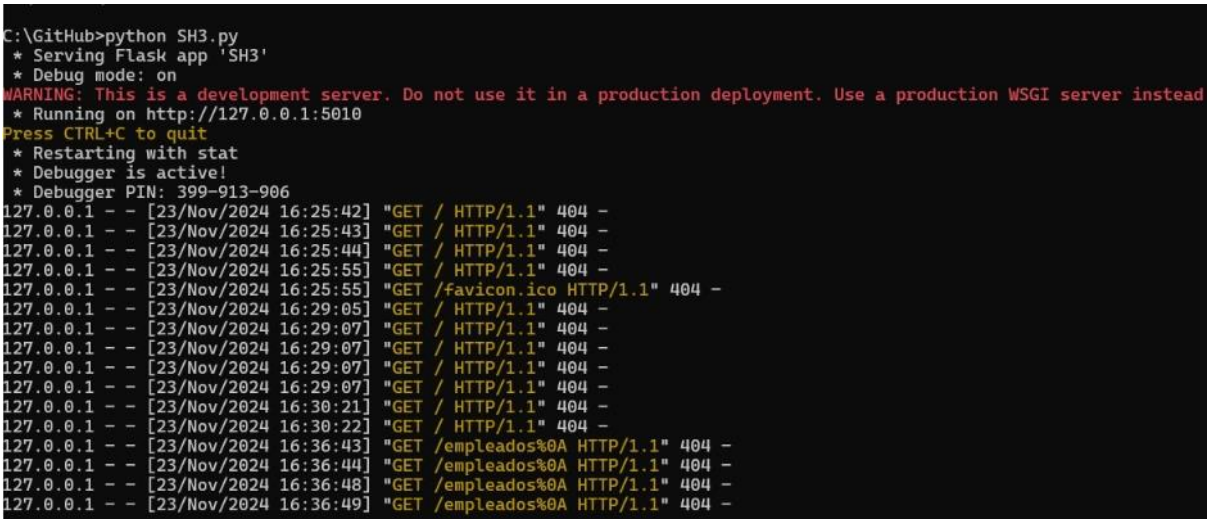
Online
Find and replace
Console
All Logs
Clear
DELETE http://127.0.0.1:5010/alumnos/2
200
232 ms
Network
Show raw log
Request Headers
Content-Type: "application/json"
User-Agent: "PostmanRuntime/7.42.0"
Accept: "*/*"
Postman-Token: "68067460-ea38-4c29-ade7-7e660c773ed3"
Host: "127.0.0.1:5010"
Accept-Encoding: "gzip, deflate, br"
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

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 (`ImportError`).	Diagnóstico: Librerías necesarias como `flask_sqlalchemy` o `pymysql` no están instaladas. Solución: Instalar dependencias con `pip install flask_sqlalchemy pymysql`.
500 Internal Server Error	Uso incorrecto de `_name_` en lugar de `__name__`.	Diagnóstico: `_name_` está mal definido en `app = Flask(_name_)`. Solución: Reemplazar `_name_` por `__name__` en el constructor de Flask.
500 Internal Server Error	Configuración incorrecta de `SQLALCHEMY_DATABASE_URI`.	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 (`/empleados`, `/empleados/<int:id>`).
400 Bad Request	Error al procesar datos JSON.	Diagnóstico: El cuerpo de la solicitud no incluye los campos esperados (`nombre`, `carrera`,

		`numero`). Solución: Validar los datos enviados por el cliente antes de procesarlos.
404 Not Found	Empleado no encontrado en una operación de `PUT` o `DELETE`.	Diagnóstico: El ID proporcionado no existe en la base de datos. Solución: Validar la existencia del empleado antes de intentar modificarlo o eliminarlo.
400 Bad Request	Violación de la restricción de unicidad (`IntegrityError`) en la columna `numero`.	Diagnóstico: Se intenta agregar o actualizar un registro con un número duplicado. Solución: Implementar validaciones para evitar duplicados antes de guardar los datos.
500 Internal Server Error	Error en la declaración del modelo SQLAlchemy (`_tablename_` en lugar de `__tablename__`).	Diagnóstico: SQLAlchemy no reconoce el atributo `_tablename_`. Solución: Cambiar `_tablename_` por `__tablename__` en la definición del modelo `Empleado`.
500 Internal Server Error	Diferencias entre el modelo SQLAlchemy y la estructura de la tabla en la base de datos.	Diagnóstico: Las columnas del modelo no coinciden con las definidas en la tabla de la base de datos (`tblEmpleados`). Solución: Asegurarse de que las columnas y sus tipos en la base de datos coincidan con las definiciones del modelo SQLAlchemy.
500 Internal Server Error	Error de tipos al manejar valores (`OperationalError` o `DataError`).	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 `to_dict` devuelve tipos de datos no serializables por JSON. Solución: Asegurar que `to_dict` solo devuelva valores como `str`, `int`, o `float`, 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.