INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

Sistemas Distribuidos

Profesor: Chadwick Carreto Arrellano

7CM1

Práctica 6 “Servicios WEB”

Juan Fernando León Medellín

ANTECEDENTE

1. API (Interfaz de Programación de Aplicaciones)

Una **API (Application Programming Interface)** es un conjunto de reglas, protocolos y herramientas que permiten que diferentes aplicaciones se comuniquen entre sí. Una API define cómo los sistemas pueden interactuar, especificando las solicitudes que pueden ser realizadas y las respuestas que se esperan.

En el desarrollo de sistemas distribuidos, las APIs son fundamentales porque permiten que las aplicaciones se integren con otros servicios o componentes de manera modular. Existen diferentes estilos o arquitecturas para diseñar APIs, siendo **REST (Representational State Transfer)** y **SOAP (Simple Object Access Protocol)** dos de los más utilizados.

2. REST (Representational State Transfer)

REST es un estilo arquitectónico para la creación de servicios web que se basa en principios sencillos y utiliza HTTP como protocolo de comunicación. REST se enfoca en los recursos, los cuales se identifican mediante URLs, y en las operaciones que se pueden realizar sobre esos recursos utilizando los métodos HTTP: **GET**, **POST**, **PUT**, **DELETE**.

Algunas características clave de las API RESTful son:

- **Ligereza y escalabilidad**: Las solicitudes y respuestas se manejan generalmente en formato **JSON** o **XML**, lo que las hace rápidas y fáciles de procesar.
- **Sin estado**: Cada solicitud contiene toda la información necesaria para procesar la petición, lo que significa que el servidor no mantiene el estado de la sesión entre diferentes solicitudes.
- **Interoperabilidad**: Las API RESTful son independientes de la plataforma y del lenguaje de programación, lo que las hace adecuadas para aplicaciones que requieren integrarse con diferentes tecnologías.

3. SOAP (Simple Object Access Protocol)

SOAP es un protocolo de mensajería que define un conjunto de reglas estrictas para el intercambio de información estructurada entre sistemas. SOAP utiliza XML para definir su estructura de mensajes y suele emplearse en entornos donde la seguridad, confiabilidad y transacciones son críticas, como en los servicios bancarios o empresariales.

Algunas características clave de SOAP son:

- **Estandarización**: SOAP está basado en un conjunto de estándares bien definidos (como WS-Security para seguridad, WS-ReliableMessaging para confiabilidad, etc.).
- **Mayor complejidad**: Aunque SOAP es más seguro y fiable, es más complejo de implementar que REST, debido a su formato XML y la necesidad de cumplir con estrictas reglas de comunicación.
- **Protocolo independiente**: SOAP puede funcionar sobre diferentes protocolos de transporte como HTTP, SMTP, TCP, entre otros.

4. Métodos HTTP: GET y POST

En el contexto de las APIs RESTful, los métodos **GET** y **POST** son dos de los más utilizados. Cada uno cumple con un propósito diferente en la interacción con el servidor.

- **GET**: El método GET se utiliza para **obtener datos** del servidor. Las solicitudes GET son idempotentes, lo que significa que obtener los mismos datos varias veces no afecta el estado del servidor. Los datos solicitados generalmente se envían como parámetros en la URL.
- **POST**: El método POST se utiliza para **enviar datos al servidor** para su procesamiento o creación. A diferencia de GET, las solicitudes POST no son idempotentes, lo que significa que pueden cambiar el estado del servidor al procesar los datos. Los datos enviados generalmente se incluyen en el cuerpo de la solicitud (body) en formato JSON, XML o formulario.

5. Postman como Herramienta para Probar APIs

Postman es una herramienta de desarrollo utilizada para **probar APIs**. Permite a los desarrolladores realizar solicitudes a las APIs, ver las respuestas y analizar el comportamiento de los servicios web. Postman facilita la creación, ejecución y documentación de las pruebas para asegurar que las APIs respondan correctamente.

Entre las funciones que Postman ofrece están:

- La posibilidad de probar diferentes métodos HTTP (GET, POST, PUT, DELETE, etc.).
- La capacidad de enviar parámetros y datos en el cuerpo de la solicitud.
- La visualización de respuestas en varios formatos (JSON, XML, HTML).

PLANTEAMIENTO DEL PROBLEMA

Desarrollar un servicio web RESTful en Python, utilizando el framework **Flask**, que permita la gestión de información relacionada con el clima de diferentes ciudades. El servicio debe soportar operaciones básicas de consulta (**GET**), inserción (**POST**) y eliminación (**DELETE**) de datos. Además, se debe emplear la herramienta **Postman** para realizar pruebas y asegurar que las respuestas del servidor sean correctas y eficientes.

Planteamientos Específicos del Problema

1. **Diseñar una API RESTful eficiente** que cumpla con los principios de diseño de REST, asegurando que el servicio sea sencillo y escalable.
2. **Implementar operaciones básicas** (GET, POST, DELETE) que gestionen datos en formato JSON, permitiendo agregar, consultar y eliminar información de manera efectiva.
3. **Realizar pruebas exhaustivas** utilizando herramientas como Postman, verificando que las solicitudes y respuestas sean correctas y el servidor funcione de acuerdo a lo esperado.
4. **Gestionar los posibles errores** y manejar las respuestas adecuadamente, asegurando que el servicio sea robusto y seguro.

MATERIALES Y MÉTODOS EMPLEADOS

Lenguaje de Programación:

- Python

Framework:

- Flask

Formato de Intercambio de Datos:

- JSON

Herramienta de Pruebas:

- Postman

PROPUESTA DE SOLUCIÓN

La solución propuesta consiste en el desarrollo de un **servicio web RESTful** que permita la gestión de información sobre el clima de diversas ciudades, utilizando el framework **Flask** en Python. Este servicio web debe ser capaz de realizar operaciones básicas de **consulta** (GET), **creación** (POST) y **eliminación** (DELETE) de datos, los cuales serán gestionados en formato **JSON**. Además, se utilizará **Postman** para realizar pruebas exhaustivas y asegurar que el servicio funciona correctamente.

1. Diseño de la API RESTful

La **API RESTful** se diseñará siguiendo los principios fundamentales de **REST**, asegurando que sea simple, escalable y eficiente. Las operaciones principales serán:

- **GET**: Para obtener la información del clima de una ciudad específica. La solicitud se realizará mediante la URL de la ciudad.
- **POST**: Para agregar datos del clima para una ciudad. La información se enviará en formato JSON en el cuerpo de la solicitud.
- **DELETE**: Para eliminar los datos del clima de una ciudad específica.

La estructura básica de la API será la siguiente:

- **Endpoint GET**: /clima/<ciudad>
    - Solicita la información del clima de la ciudad especificada.
- **Endpoint POST**: /clima
    - Permite agregar información del clima para una nueva ciudad.
- **Endpoint DELETE**: /clima/<ciudad>
    - Elimina la información del clima de la ciudad especificada.

2. Implementación con Flask

Se utilizará **Flask**, un framework ligero de Python, para desarrollar el servicio web. Flask es adecuado para este tipo de proyectos porque permite crear APIs de manera rápida y sencilla, y proporciona flexibilidad para manejar rutas y métodos HTTP de manera eficiente.

3. Pruebas con Postman

Una vez implementado el servicio, se utilizará **Postman** para probar los diferentes endpoints y verificar su funcionamiento.

- **Prueba GET**: Enviando una solicitud GET a /clima/<ciudad>, donde <ciudad> es el nombre de la ciudad que queremos consultar. El servicio debe devolver la información del clima en formato JSON.
- **Prueba POST**: Enviando una solicitud POST a /clima, con el cuerpo de la solicitud en formato JSON que contenga la ciudad, temperatura y humedad. El servidor debe agregar estos datos a la base de datos simulada y devolver un mensaje de éxito.
- **Prueba DELETE**: Enviando una solicitud DELETE a /clima/<ciudad>, donde <ciudad> es el nombre de la ciudad cuyo clima queremos eliminar. El servidor debe eliminar los datos y devolver un mensaje de éxito.

4. Pruebas Automáticas

Se puede utilizar **Postman Collections** para organizar las pruebas y automatizarlas. Las colecciones permiten realizar un conjunto de pruebas con diferentes datos, lo que facilita la comprobación de que el servicio funciona correctamente bajo distintos escenarios.

5. Manejando Errores y Respuestas

La API debe manejar adecuadamente los errores, como cuando se solicitan datos de una ciudad que no existe o cuando los datos enviados en una solicitud POST están incompletos. Las respuestas serán claras y se devolverán códigos de estado HTTP apropiados, como:

- **200 OK**: La solicitud fue exitosa.
- **201 Created**: El recurso fue creado correctamente.
- **400 Bad Request**: La solicitud tiene errores (por ejemplo, falta información).
- **404 Not Found**: El recurso no fue encontrado (por ejemplo, la ciudad no existe).
- **500 Internal Server Error**: Error inesperado en el servido

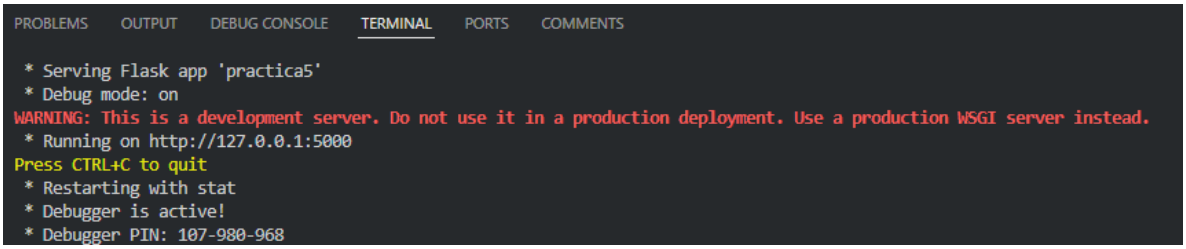
DESARROLLO DE LA SOLUCIÓN

En el siguiente enlace se puede encontrar el código de la práctica:

https://github.com/JFernandoLe/SD_PRACTICA6.git

RESULTADOS

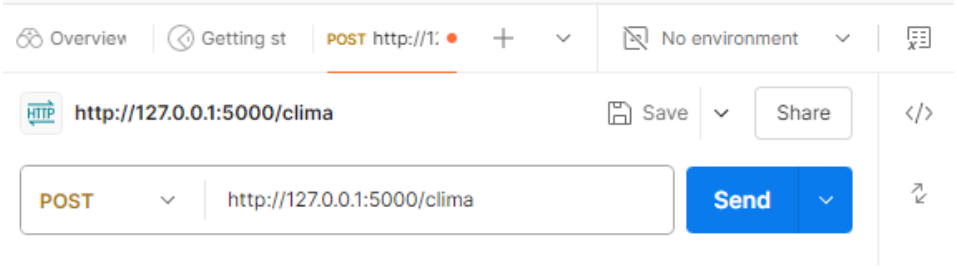
Servidor. Inicializamos el servidor en http://127.0.0.1:5000



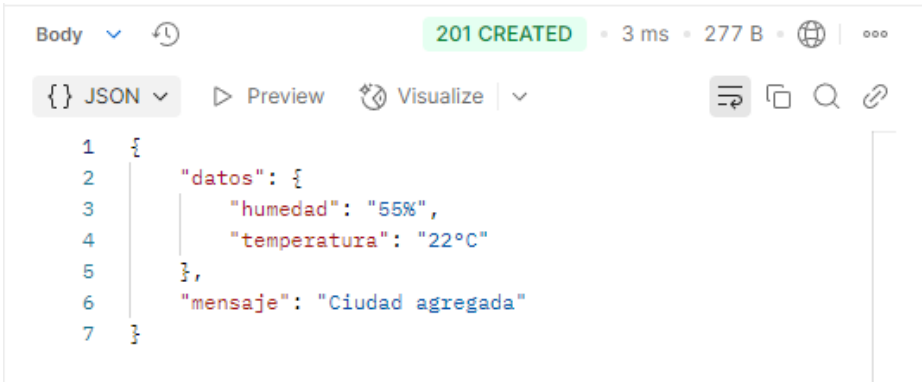
Pruebas con Postman. Consultamos el clima de una ciudad (GET)

Respuesta JSON

Agregar una nueva ciudad (POST)



Respuesta JSON



Eliminar una ciudad (DELETE)

Respuesta JSON

CONCLUSIÓN

La práctica realizada ha permitido comprender y aplicar los conceptos fundamentales de la creación y prueba de servicios web utilizando el estilo arquitectónico **REST**. A lo largo del proceso, se desarrolló un servicio web básico con **Flask** en Python que permite gestionar datos de manera sencilla, realizando operaciones de **consulta** (GET), **creación** (POST) y **eliminación** (DELETE) de información, todo gestionado mediante formato **JSON**.

El uso de **Postman** ha sido clave para probar y verificar el correcto funcionamiento de la API, asegurando que las solicitudes y respuestas sean procesadas adecuadamente por el servidor. Las pruebas realizadas mediante **GET**, **POST** y **DELETE** demostraron que el servicio responde correctamente a las solicitudes y maneja adecuadamente los posibles errores.

Al mismo tiempo, la práctica ha permitido reflexionar sobre las diferencias entre los estilos arquitectónicos **REST** y **SOAP**, destacando la simplicidad y flexibilidad de REST para desarrollar aplicaciones ligeras y escalables. Además, la correcta implementación de los métodos HTTP y el manejo de datos en formato JSON resulta en un servicio eficiente y fácil de integrar con otros sistemas.

En resumen, esta práctica ha logrado el objetivo de ofrecer una solución funcional para la gestión de información mediante un servicio web RESTful, proporcionando una base sólida para el desarrollo y prueba de APIs en sistemas distribuidos. La experiencia adquirida en la creación, implementación y prueba de un servicio web es fundamental para afrontar proyectos más complejos en el futuro, donde se requiera integrar diferentes componentes o servicios.