CampusDual TIC

AI Engineering

Clase 3 - Requests, API and Auth

Introducción a Python - Requests, Autenticación y APIs para Tareas de Generative AI

Duración: 45 minutos

Contexto:

- Introducción a una sesión teórica enfocada en el uso de la librería `requests` en Python para interactuar con APIs externas.
- Enfatiza la importancia de estos conceptos en el desarrollo de aplicaciones de **Inteligencia Artificial Generativa (GenAI)**.

Contexto:

- **Relevancia de las APIs:** Son fundamentales para acceder a modelos de lenguaje y otros recursos externos en GenAI.
- **Uso de `requests`:** Simplifica la interacción con APIs, permitiendo desarrollos más eficientes y robustos.

1. Componentes Clave de HTTP

- **Métodos HTTP:**
  - **GET:** Solicita datos de un recurso específico.
  - **POST:** Envía datos al servidor para crear o actualizar un recurso.
  - **PUT:** Actualiza un recurso existente.
  - **DELETE:** Elimina un recurso específico.
- **URLs y Endpoints:**
  - **URL (Uniform Resource Locator):** Dirección utilizada para acceder a recursos en la web.
  - **Endpoint:** Punto final de comunicación en una API donde se pueden realizar solicitudes.

2. Introducción a la Librería `requests` en Python

¿Por qué `requests`?

Definición: La librería `requests` es una de las más populares y sencillas de usar para realizar solicitudes HTTP en Python. Proporciona una interfaz amigable para interactuar con APIs sin necesidad de manejar los detalles de bajo nivel del protocolo HTTP.

```python
import requests


# Realizar una solicitud GET
response = requests.get('https://api.example.com/data')


# Verificar el estado de la respuesta
if response.status_code == 200:
    datos = response.json()
    print(datos)
else:
    print(f"Error: {response.status_code}")
```

3. Realización de Solicitudes HTTP

Solicitudes GET

Definición: Utilizadas para solicitar datos de un servidor. No deben modificar el estado del recurso.

Ejemplo: Obtener una lista de APIs públicas

```
import requests

def obtener_apis_publicas():
    url = "https://api.publicapis.org/entries"
    response = requests.get(url)
    if response.status_code == 200:
        return response.json()
    else:
        print(f"Error: {response.status_code}")
        return None

datos = obtener_apis_publicas()
if datos:
    print(f"Total de APIs disponibles: {len(datos['entries'])}")
```

Solicitudes POST

Definición: Utilizadas para enviar datos al servidor, por ejemplo, para crear un nuevo recurso.

Ejemplo: Enviar datos a una API de prueba

```python
import requests

def enviar_datos_api():
    url = "https://httpbin.org/post"
    payload = {"clave": "valor"}
    headers = {"Content-Type": "application/json"}
    response = requests.post(url, json=payload, headers=headers)
    if response.status_code == 200:
        return response.json()
    else:
        print(f"Error: {response.status_code}")
        return None

respuesta = enviar_datos_api()
if respuesta:
    print("Datos enviados exitosamente:")
    print(respuesta)
```

4. Autenticación en APIs

¿Por qué es importante la autenticación?

Definición: La autenticación asegura que solo usuarios autorizados puedan acceder a los recursos de una API. Protege los datos y garantiza que las solicitudes sean legítimas.

Contexto:

- **Seguridad de los datos:** Protege información sensible y recursos críticos.
- **Control de acceso:** Asegura que solo usuarios autorizados puedan interactuar con las APIs, evitando abusos y accesos no deseados.

Métodos Comunes de Autenticación

- **API Keys:** Claves únicas proporcionadas por la API para identificar al usuario.
- **Bearer Tokens:** Tokens de acceso que se incluyen en los encabezados de las solicitudes.
- **OAuth:** Protocolo de autorización que permite a las aplicaciones acceder a los recursos de los usuarios sin exponer sus credenciales.

Implementación de API Keys

Ejemplo: Autenticación con la API de OpenAI

```python
import requests

class ModeloOpenAI:
    def __init__(self, api_key):
        self.api_key = api_key
        self.url = "https://api.openai.com/v1/completions"
        self.headers = {
            "Authorization": f"Bearer {self.api_key}",
            "Content-Type": "application/json"
        }

    def generar_texto(self, prompt, modelo="text-davinci-003", max_tokens=150):
        payload = {
            "model": modelo,
            "prompt": prompt,
            "max_tokens": max_tokens,
            "temperature": 0.7
        }
        response = requests.post(self.url, headers=self.headers, json=payload)
        if response.status_code == 200:
            return response.json()["choices"][0]["text"].strip()
        else:
            print(f"Error: {response.status_code} - {response.text}")
            return None

if __name__ == "__main__":
    api_key = "tu_api_key_aquí"  # Reemplaza con tu clave API de OpenAI
    modelo_openai = ModeloOpenAI(api_key)
    prompt = "Escribe un poema sobre la inteligencia artificial."
    texto_generado = modelo_openai.generar_texto(prompt)
    if texto_generado:
        print("Texto Generado por OpenAI:")
        print(texto_generado)
```

Manejo de Excepciones

Definición: Utiliza bloques `try-except` para capturar y manejar excepciones que puedan ocurrir durante las solicitudes.

```python
import requests

def obtener_datos_con_excepcion(url):
    try:
        response = requests.get(url)
        response.raise_for_status()  # Lanza una excepción para códigos de estado 4xx/5xx
        return response.json()
    except requests.exceptions.HTTPError as http_err:
        print(f"HTTP error occurred: {http_err}")
    except requests.exceptions.ConnectionError as conn_err:
        print(f"Error de conexión: {conn_err}")
    except requests.exceptions.Timeout as timeout_err:
        print(f"Tiempo de espera agotado: {timeout_err}")
    except requests.exceptions.RequestException as req_err:
        print(f"Error inesperado: {req_err}")
    return None
```

6. Integración con Generative AI

Caso de Estudio: API de OpenAI para Generación de Textos

Objetivo: Integrar la API de OpenAI en una aplicación para generar textos basados en prompts proporcionados por el usuario.

Contexto:

- **Aplicación práctica de los conceptos:** Muestra cómo utilizar `requests`, autenticación y manejo de respuestas en un contexto real de GenAI.
- **Demostración de funcionalidad:** Permite a los participantes ver cómo se integra una API avanzada en una aplicación Python.

```python
import requests

class ModeloOpenAI:
    def __init__(self, api_key):
        self.api_key = api_key
        self.url = "https://api.openai.com/v1/completions"
        self.headers = {
            "Authorization": f"Bearer {self.api_key}",
            "Content-Type": "application/json"
        }

    def generar_texto(self, prompt, modelo="text-davinci-003", max_tokens=150):
        payload = {
            "model": modelo,
            "prompt": prompt,
            "max_tokens": max_tokens,
            "temperature": 0.7
        }
        try:
            response = requests.post(self.url, headers=self.headers, json=payload)
            response.raise_for_status()
            return response.json()["choices"][0]["text"].strip()
        except requests.exceptions.HTTPError as http_err:
            print(f"HTTP error occurred: {http_err}")
        except requests.exceptions.RequestException as req_err:
            print(f"Error en la solicitud: {req_err}")
        return None

if __name__ == "__main__":
    api_key = "tu_api_key_aquí"  # Reemplaza con tu clave API de OpenAI
    modelo_openai = ModeloOpenAI(api_key)
    prompt = "Escribe un cuento sobre un robot que aprende a sentir emociones."
    texto_generado = modelo_openai.generar_texto(prompt)
    if texto_generado:
        print("Texto Generado por OpenAI:")
        print(texto_generado)
```

7. Buenas Prácticas

Manejo Seguro de Claves API

- No expongas tus claves API en el código fuente.
- Utiliza variables de entorno o archivos de configuración seguros.

```
import os


api_key = os.getenv("OPENAI_API_KEY")
```

PREGUNTAS

CampusDual TIC