API-Doc-Scraper.py

import asyncio

import aiohttp

import requests

import re

import os

import time

from urllib.parse import urlparse

from openai import OpenAI

from dotenv import load\_dotenv

# Load environment variables from .env file

load\_dotenv()

def get\_user\_url():

url = input("Please enter a URL for the documentation: ").strip()

if not url.startswith(('http://', 'https://')):

url = 'https://' + url

return url

def is\_valid\_url(url):

try:

result = urlparse(url)

return all([result.scheme, result.netloc])

except ValueError:

return False

async def scrape\_url\_async(session, url, max\_retries=3):

jina\_api\_url = f"https://r.jina.ai/{url}"

headers = {

"Authorization": f"Bearer {os.getenv('JINA\_API\_KEY')}",

"X-Timeout": "10"

}

for attempt in range(max\_retries):

try:

async with session.get(jina\_api\_url, headers=headers) as response:

response.raise\_for\_status()

return await response.text()

except aiohttp.ClientError as e:

if attempt == max\_retries - 1:

print(f"Error scraping URL after {max\_retries} attempts: {e}")

return None

await asyncio.sleep(2 \*\* attempt) # Exponential backoff

async def scrape\_urls\_concurrently(urls):

async with aiohttp.ClientSession() as session:

tasks = [scrape\_url\_async(session, url) for url in urls]

return await asyncio.gather(\*tasks)

def extract\_urls(content):

url\_pattern = r'https?://[^\s)\]"]+'

urls = re.findall(url\_pattern, content)

unique\_urls = []

seen = set()

for url in urls:

if url not in seen and is\_valid\_url(url):

seen.add(url)

unique\_urls.append(url)

return unique\_urls

def get\_filename\_from\_url(url):

parsed\_url = urlparse(url)

domain\_parts = parsed\_url.netloc.split('.')

if len(domain\_parts) > 2 and domain\_parts[0] != 'www':

domain = f"{domain\_parts[0]}\_{domain\_parts[1]}"

elif domain\_parts[0] == 'www':

domain = domain\_parts[1]

else:

domain = domain\_parts[0]

return f"{domain}\_docs.txt"

def create\_content\_section(content, url, index):

separator = "=" \* 80

return f"\n{separator}\nSection {index}: Content from {url}\n{separator}\n\n{content}\n\n"

def write\_content\_to\_file(filename, sections):

with open(filename, 'w', encoding='utf-8') as file:

file.write("".join(sections))

def filter\_urls(urls, base\_url):

client = OpenAI(api\_key=os.getenv('OPENAI\_API\_KEY'))

urls\_text = "\n".join(urls)

base\_domain = urlparse(base\_url).netloc

response = client.chat.completions.create(

model="gpt-4o-mini",

messages=[

{"role": "system", "content": "You are a URL curator tasked with filtering out obviously unrelated content from a list of URLs for a software tool or API."},

{"role": "user", "content": f"""We need to extract documentation for a tool with the base domain {base\_domain}. Here's a list of URLs we've found:

{urls\_text}

Please filter this list based on the following criteria:

1. Keep URLs that appear to be related to documentation, guides, tutorials, or API references.

2. Include relevant subdomains like 'docs.{base\_domain}', 'api.{base\_domain}', or 'developer.{base\_domain}'.

3. Remove URLs for obviously unrelated content such as community forums, status pages, blog posts, or contact pages.

Respond with a list of filtered URLs, one per line, without any additional text or formatting. Ensure all URLs are valid."""}

]

)

filtered\_urls = [url.strip() for url in response.choices[0].message.content.strip().split('\n') if is\_valid\_url(url.strip())]

return filtered\_urls

async def main\_async():

user\_url = get\_user\_url()

print(f"You entered: {user\_url}")

async with aiohttp.ClientSession() as session:

initial\_content = await scrape\_url\_async(session, user\_url)

if initial\_content:

urls = extract\_urls(initial\_content)

print(f"Found {len(urls)} unique URLs.")

filtered\_urls = filter\_urls(urls, user\_url)

print(f"Filtered down to {len(filtered\_urls)} relevant URLs.")

filename = get\_filename\_from\_url(user\_url)

contents = await scrape\_urls\_concurrently(filtered\_urls)

sections = []

for i, (url, content) in enumerate(zip(filtered\_urls, contents), 1):

if content:

sections.append(create\_content\_section(content, url, i))

print(f"URL #{i} Scraped ✅ - {url}")

else:

print(f"URL #{i} Found an Error and Skipped ❌ - {url}")

write\_content\_to\_file(filename, sections)

print(f"📃 Content saved to {filename}. 📃")

else:

print("❌ Failed to scrape the initial URL. ❌")

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.run(main\_async())