Handleiding - Het bouwen van een eigen medische chatbot
E.R.Rozenveld
versie 1.0

Inleiding

Deze handleiding bevat de bouw voor een dataspecifieke chatbot waarin websites, of eigen documenten gebruikt kunnen worden om een chatbot aan te sturen.

een voorbeeld:

```
Welkom bij PharmAssist! Stel je medische vraag of typ 'exit' om te stoppen.
>> Wat zijn de farmacokinetische verschillen tussen de orale toediening van levothyroxine en liothyronine bij patiënten met hypothyreoïdie?
Analyseren van vraag...
Uitvoeren van zoekopdracht: farmacokinetische verschillen orale toediening levothyroxine liothyronine hypothyreoïdie
Gevonden resultaten:
- levothyroxine: https://www.farmacotherapeutischkompas.nl/bladeren/preparaatteksten/l/levothyroxine
Gescraapte tekst van https://www.farmacotherapeutischkompas.nl/bladeren/preparaatteksten/l/levothyroxine: U bevindt zich hier:

levothyroxine vergelijken met een ander geneesmiddel.
 Bij een niet-iatrogene primaire hypothyreoïdie (verhoogd TSH en verlaagd vrije T4) is de standaardbehandeling suppletie met levothyroxine. Bij (sub)acute thyroïditis kan levothyroxine alleen kortdurend overwogen worden bij sy
ptomatische hypothyreoïdie. Bij een subklinische hypothyreoïdie (verhoogd TSH en normaal vrije T4) kan, afhankelijk van de TSH-waarde, leeftijd en eventuele zwangerschap, levothyroxine in bepaalde gevallen overwogen worden. Bi
hyperthyreoïdie door de ziekte van Graves heeft medicamenteuze behandeling met carbimazol of thiamazol, met of zonder levothyroxine, in eerste instantie de voorkeur. Bij een recidief zijn de drie bovengenoemde opties gelijkwa
dig. Bij veel klachten kan in de eerste weken ter overbrugging (tot het bereiken van euthyreoïdie) een β-blokker worden toegevoegd, bv. atenolol (offlabel), metoprolol succinaat met gereguleerde afgifte of propranolol. De keu
 voor welke β-blokker is afhankelijk van de comorbiditeit. Stel de individuele dagelijkse dosis vast op geleide van de klinische respons en laboratoriumtesten (TSH en vrij T4). Volgens de behandelrichtlijnen bij hypothyreoïdi
de serumhormoonspiegels niet eerder dan 6 weken na elke dosisverandering beoordelen, aangezien het TSH dan pas is gestabiliseerd.  Let op! Vermijd het wisselen van levothyroxinepreparaten. Indien wisseling noodzakelijk is va
ege het niet beschikbaar zijn van preparaten, en er klachten ontstaan, controleer het TSH dan ten minste 6 weken na wisseling.
```

Antwoord + bronvermelding:

```
Antwoord: Beste Apotheker,

De farmacokinetische verschillen tussen de orale toediening van levothyroxine en liothyronine bij patiënten met hypothyreoïdie zijn als volgt:

- Levothyroxine is de standaardbehandeling voor hypothyreoïdie en wordt voornamelijk gebruikt voor de suppletie van schildklierhormonen. Liothyronine wordt minder vaak gebruikt en wordt
g van (sub)acute thyroïditis.
- Levothyroxine heeft een langere halfwaardetijd dan liothyronine, wat betekent dat het langer in het lichaam blijft en minder vaak hoeft te worden ingenomen.
- Levothyroxine wordt in de lever omgezet in het actieve schildklierhormoon T3, terwijl liothyronine al in zijn actieve vorm is. Dit kan leiden tot een snellere werking van liothyronine
e in het bloed.
>>
```

```
Bronnen:
- Gevonden op https://www.farmacotherapeutischkompas.nl/bladeren/preparaattekster
- Gevonden op https://www.farmacotherapeutischkompas.nl/bladeren/preparaattek
```

Omdat de chatbot alleen informatie mag geven van bronnen die die kan vinden, zal die niet snel iets gaan ‘verzinnen’

```
>> Beste voetbalclub van de wereld?
Analyseren van vraag...
Uitvoeren van zoekopdracht: beste voetbalclub wereld
Geen relevante resultaten gevonden.
>>
```

LangChain Framework

Het framework van de chatbot bestaat uit LangChain.
dit framework biedt al de tools (PromptTemplate, DocumentSplitter, Documentloaders, etc.) voor een snelle start
We zullen dan ook alle imports uit LangChain halen, voor meer informatie: Zie de LangChain handleiding

Handleiding - Het bouwen van een eigen medische chatbot
E.R.Rozenveld
versie 1.0

CSE-API & CSE-ID

De chatbot werkt verder met een custom-search-engine, in dit voorbeeld is Google gebruikt omdat het een snelle efficiënte zoekmachine is. Echter kan je ook andere CSE gebruiken of zelfs de webpaginas hardcoden.
Begin met het maken van een CSE: Zie handleiding
Tip: met het gebruik van asteriks (*) in de URLs geef je aan op welke pagina's er gezocht kan worden.

Let op: er zitten wel limitaties aan de CSE in de gratis versie, er kan niet veel verkeer/request door, deze limitatie is NIET hard-coded in de huidige chatbot waardoor als er instellingen wijzigen er een foutcode kan ontstaan door teveel requests, dit kan opgelost worden door een time-limit van request in te bouwen (iets langere zoektijd) of te betalen voor meer limiet.

Open-AI-API

Maak een Open-AI API-key aan: Zie handleiding
Ook voor de OpenAI-API geldt een limiet van gebruikersverkeer. Er kunnen GEEN enorme lappen tekst door de bot verwerkt worden. Dit is opgelost met de TekstSplitter (dus chunks te voeren aan de chatbot) en een limiet van tekens (wanneer je de uservraag combineert met de informatie) >>>>>> dit is nog niet juist gebouwd in de huidige versie omdat ik niet wil dat de uservraag met maar een chunk van de informatie een antwoord formuleert, hier is wel een oplossing voor denk ik om de chunks afzonderlijk weer naar de chatbot te sturen.

Set API in environment

Voeg de API's toe in de omgevingsvariabele:
setx OPENAI_API_KEY "jouw_openai_api_key"
setx GOOGLE_API_KEY "jouw_google_api_key"
setx CSE_ID "jouw_cse_id"

Bovenstaande is anders in een Maccomputer! volgens mij gebruik je daar export, ipv setx

Open PyCharm of Visuals (ik gebruik Visuals)
In de terminal heb je de volgende paketten nodig:
pip install requests
pip install beautifulsoup4
pip install langchain langchain-openai
pip install google-api-python-client

>>>>pip install pypdf voor de pdf upload functie.

request is nodig om naar de websites te navigeren
beautifulsoup4 is een scraping functie die stukken tekst uit de website haalt

Onderstaande zijn de imports die gebruikt zijn

```
import requests
from bs4 import BeautifulSoup
from langchain.llms import OpenAI
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.document_loaders import TextLoader, PyPDFLoader
from langchain.schema import Document
from googleapiclient.discovery import build
import os
```

De bouw:

Allereerst moet de Uservraag worden omgezet tot een leesbare variant voor de CSE. Het kopiëren van de gehele vraag werkt niet, maar met steekwoorden werkt het wel:

```
def extract_keywords_with_chatgpt(user_input):
    """Gebruik ChatGPT om relevante steekwoorden te extraheren."""
    llm = OpenAI(temperature=0, openai_api_key=OPENAI_API_KEY)
    prompt = PromptTemplate(
        input_variables=["vraag"],
        template="Geef een lijst van maximaal 6 steekwoorden voor de
volgende zoekopdracht, geef alleen de steekwoorden zonder nummers,
punten of andere tekens, gescheiden door spaties: {vraag}"
    )
    chain = LLMChain(llm=llm, prompt=prompt)
    response = chain.run({"vraag": user_input})
```

Dit laat direct de kracht van LangChain zien doordat ik hier al gebruik maak van de PromptTemplate functie. Hierin geef ik eigenlijk de opdracht uit, getypt wat er moet gebeuren met de Uservraag. **Hierin kun je dus al veel tweaken.** Je kan bijvoorbeeld aangeven dat er op combinaties van woorden gezocht moet worden, of altijd in context van een bepaalde ziekte, noem maar op. Met de ‘ temperature’ functie beheer je de vrijheid die de chatbot mag nemen met het interpeteren van de data, deze heb ik op 0 gezet om zoveel mogelijk de feiten te gebruiken.

Daarna gaat het de CSE in:

```
def google_search(query, num_results=10):
```

```
    service = build("customsearch", "v1", developerKey=GOOGLE_API_KEY)
    search_results = []
    try:

        result = service.cse().list(q=query, cx=CSE_ID,
num=num_results).execute()
        search_results.extend(result.get("items", []))
    except Exception as e:
        print(f"Fout bij zoeken: {e}")
    return search_results
```

Hierin kun je de resultaten met num_results=X tweaken. Huidig is er een limitatie in de hoeveelheid webpaginas die gevonden wordt (significant sneller), maar dit kan aangepast worden (trager> meer resultaten > meer verwerking voor ChatGPT)

```
def split_text(text):
    """Splits de tekst in kleinere stukken voor verwerking."""
    splitter = RecursiveCharacterTextSplitter(chunk_size=800,
chunk_overlap=100)
    return splitter.split_text(text)
```

Tekst splitsing , dit is een belangrijke parameter, voor verwerking de juiste balans nodig tussen de **size**, en de **overlap.** Je wilt niet te laag sizen, omdat je dan belangrijke informatie kan missen. Echter, te grote chunks en je verwerking wordt trager en je bereikt sneller je limiet...

```
def generate_answer(text_chunks, user_input):
    """Genereer een antwoord op basis van de samengevoegde tekst."""
    llm = OpenAI(temperature=0, openai_api_key=OPENAI_API_KEY)
    prompt = PromptTemplate(
        input_variables=["informatie", "vraag"],
        template="""Je bent een deskundige ziekenhuisapotheker. Geef
een medisch onderbouwd antwoord.
        Vraag: {vraag}
        Informatie: {informatie}
        """
```

Bovenstaande is het hart van de code, wederom met de PromptTemplate Functie. Hier komt de informatie en de Uservraag samen. De template kun je op iedere manier die je wilt inrichten met tekstuele commando's. Je kan bijvoorbeeld aangeven dat de chatbot altijd moet antwoorden met ‘ Beste Apotheker’ . Het werkt ook om dat prompt zeer specifiek te maken voor zeer specifieke taken, echter kan het (door de input van de informatie) wel conflicterende resultaten geven.

Ik heb tijdens testen gezien dat sommige prompts beter werken dan anderen. In dit voorbeeld is het een zeer simpele Prompt, maar deze kun je uiteraard naar wens uitwerken, bijvoorbeeld:

Je bent een deskundige ziekenhuisapotheker met uitgebreide kennis van oncologie, farmacologie, therapieën, en medische richtlijnen. Je antwoordt kort en direct. Je hebt toegang tot de meest actuele medische literatuur en richtlijnen en kunt complexe medische vragen beantwoorden vanuit een wetenschappelijk onderbouwd perspectief.

***Vraag:** {vraag}*

***Informatie:** {informatie}*

Beantwoord de vraag op basis van de onderstaande informatie, waarbij je zorgt voor een gedegen wetenschappelijke onderbouwing. Maak gebruik van relevante richtlijnen, farmaceutische inzichten en recente onderzoeken om een volledig en goed onderbouwd antwoord te geven. Geef daarbij duidelijk aan of er aanvullende informatie nodig is om de vraag volledig te beantwoorden of om een specifieke therapeutische keuze te maken. Zorg ervoor dat je antwoord duidelijk en goed gestructureerd is, met een focus op de meest effectieve behandelingsopties, medicijninteracties, mogelijke bijwerkingen en risicofactoren.

Misschien werkt zoiets ook:

***WEL:** Gebruik medicatie specifieke informatie voor je antwoord.*
***WEL:** geef altijd de bronvermelding*
***NIET:** Gebruik geen veronderstellingen of onbetrouwbare bronnen. Beperk je antwoord tot de informatie die is gegeven.*
***WEL:** vraag naar relevante labwaarden*

Dit kan allemaal getest worden!!!

```
def main():
    print("Welkom bij PharmAssist! Stel je medische vraag of typ 'exit'
om te stoppen.")
    while True:
        user_input = input(">> ")
        if user_input.lower() == "exit":
            print("Bedankt voor het gebruik van PharmAssist. Tot
ziens!")
            break
        try:
            print("Analyseren van vraag...")
            keywords = extract_keywords_with_chatgpt(user_input)
            if not keywords:
                print("Geen relevante zoektermen gevonden.")
```

```
                continue

            query = " ".join(keywords)
            print(f"Uitvoeren van zoekopdracht: {query}")
            search_results = google_search(query, num_results=10)

            if not search_results:
                print("Geen relevante resultaten gevonden.")
                continue

            print("Gevonden resultaten:")
            all_text = ""
            for result in search_results:
                if len(all_text) >= 8000:
                    break  # Stop zodra we 2000 tekens hebben
                print(f"- {result.get('title')}: {result.get('link')}")
                text = scrape_website(result.get("link"))
                print(f"Gescraapte tekst van {result.get('link')}:
{text[:8000]}...")  # Laat een stukje van de gescrapete tekst zien
                all_text += text[:8000 - len(all_text)]  # Voeg toe tot
limiet is bereikt

            text_chunks = split_text(all_text)
            print("Genereren van antwoord...")
            answer = generate_answer(text_chunks, user_input)
            print(f"\nAntwoord: {answer}")
        except Exception as e:
            print(f"Er is een fout opgetreden: {e}")
```

Dit laatste stukje tekst is eigenlijk al een beetje de interface bouw om ermee te kunnen werken.
Wat belangrijk is, of wat ik zelf belangrijk vond, is om het hele proces inzichtelijk te maken, dit is handig voor testen maar eigenlijk ook niet gek voor het gebruik. Door de ‘print’ functie zie je waar de chatbot mee bezig is. In dit geval laat het de zoektermen zien, daarna de website waar de informatie gevonden is, daarna de tekst (limiet van 8000 tekens).

De huidige bouw is geoptimaliseerd voor **snelheid**. de chatbot gebruikt nog geen memory functie, dit is via LangChain nog wel in te bouwen

🚀What's Next:🚀

- Optimalisatie van de PrompTemplate functie;
- Alternatief voor webscraping ///// API-request, Selenium oid, zodat javascript-heavy websites ook geopend kunnen worden;
- Bouwen van een Interface (bijv. met Flask) zodat er niet in de terminal gewerkt hoeft te worden;
- Memory functie: ConversationChain van LangChain inbouwen.