



Filia **UwB**
w Wilnie

FILIA UNIWERSYTETU W BIAŁYMSTOKU W WILNIE
WYDZIAŁ EKONOMICZNO-INFORMATYCZNY
KIERUNEK: INFORMATYKA

MYKYTA NOVIKOV

Nr albumu: 88327

Prototyp chatbota rekomendującego książki

Praca licencjacka
napisana pod kierunkiem
prof. dr. hab. inż.
Mieczysława Muraszkiewicza

Wilno, czerwiec 2026

Pusta strona w przypadku obustronnego drukowania

STRZESZCZENIE

Niniejsza praca obejmuje utworzenie działającego prototypu chatbota o nazwie LitMatch służącego do rekomendacji książek na podstawie promptu użytkownika. LitMatch korzysta z techniki osadzeń wektorowych (*ang.* vector embeddings) do transformacji informacji o książkach w wektory oraz z generowania wspomaganego pobieraniem (*ang.* retrieval-augmented Generation, RAG) do tworzenia rekomendacji. Dany chatbot wykorzystuje wielki model językowy, który w swoich odpowiedziach odnosi się do wektorowej bazy danych będącej wiarygodnym zewnętrznym źródłem wiedzy. LitMatch pozwala użytkownikom prosić o rekomendacje w oparciu o autora, nazwę ostatnio przeczytanego utworu i jego cechy, np. tempo narracji i epokę literacką. Chatbot jest dostępny w dwóch językach: polskim i angielskim. Projekt został napisany w języku Python i korzysta z dwóch baz danych: PostgreSQL i ChromaDB. Baza danych PostgreSQL zawiera informacje o książkach pobrane z pliku Excel, wtedy jak ChromaDB jest wektorową bazą danych, która jest wykorzystywana w mechanizmie generowania rekomendacji. Wzbogacenie informacji o utworach oraz tworzenie odpowiedzi na pytanie użytkownika wykonuje wielki model językowy Llama 4 Scout, komunikacja z którym odbywa się za pomocą Groq API.

Słowa kluczowe: chatbot, wielki model językowy, rekomendacja książek, wektorowa baza danych, generowanie wspomaganie pobieraniem

ABSTRACT

This thesis involves the development of a functioning chatbot prototype named LitMatch, designed to recommend books based on user prompt. LitMatch utilizes vector embeddings to transform information about books into vectors, as well as retrieval-augmented generation to create recommendations. This chatbot employs a large language model, which relies on a vector database as a reliable external source of knowledge. LitMatch allows users to request recommendations based on an author, the title of a recently read work, and its characteristics, such as narrative pacing and literary period. The chatbot is available in two languages: Polish and English. The project is implemented in Python and uses two databases: PostgreSQL and ChromaDB. PostgreSQL database stores information about books extracted from an Excel file, while ChromaDB serves as a vector database used in the recommendation generation process. The enrichment book data and the generation of responses to user queries are performed by the large language model Llama 4 Scout, which communicates through Groq API.

Keywords: chatbot, large language model, book recommendation, vector database, retrieval-augmented generation

SPIS TREŚCI

ROZDZIAŁ 1.	MOTYWACJA I ZAKRES PRACY	9
1.1	Chatboty	9
1.2	Czytelnictwo w XXI wieku	10
1.3	Motywacja	10
1.4	Zakres pracy	11
ROZDZIAŁ 2.	KLUCZOWE POJĘCIA I TECHNOLOGIE	12
2.1	Osadzenia wektorowe	12
2.2	Wielkie modele językowe	12
2.3	Generowanie wspomagane pobieraniem	12
ROZDZIAŁ 3.	REALIZACJA	14
3.1	Plik .env	14
3.2	Plik create_table.py	14
3.3	load_data.py	16
3.4	enrich_data.py	18
3.5	book_to_vector.py	23
3.6	query_chatbot.py	27
ROZDZIAŁ 4.	TESTOWANIE	49
4.1	Rekomendacja na podstawie tytułu	49
4.2	Rekomendacja na podstawie tytułu i opisu	49
4.3	Rekomendacja na podstawie opisu	50
4.4	Rekomendacja na podstawie autora	51
4.5	Powierzchowe zapytanie	52
4.6	Utwór spoza bazy danych	52
4.7	Weryfikacja języka angielskiego	53
ZAKOŃCZENIE		55

SPIS RYSUNKÓW	57
BIBLIOGRAFIA I ŹRÓDŁA INTERNETOWE	58

WSTĘP

Czytanie książek od zawsze było ważną czynnością w życiu ludzkim. Na długo przed pojawieniem się Internetu, czytanie było jednym z głównych sposobów zdobywania wiedzy i poszerzenia horyzontów. Z upływem czasu, ludzie zrozumieli, że wpływ książek sięga o wiele lepiej niż dostarczenie informacji. Czytanie pozwala osobom, która je uprawiają, na wzbogacenie słownictwa, rozwój umiejętności wyrażania swoich myśli, wzmocnienie myślenia krytycznego i empatii, oraz odnalezienie spokoju w coraz bardziej wymagającym świecie.

Wbrew wszystkim wyżej wspomnianym i wielu innym korzyściom, popularność czytania w XXI w. ciągle spada. Wyniki badań sugerują, że zmniejsza się nie tylko liczba książek przeczytanych w ciągu roku¹, ale także liczba ludzi czytających książki dla przyjemności². Badania wskazują na rozpraszacze cyfrowe, zmniejszającą się ilość czasu wolnego oraz trudności ze znalezieniem odpowiedniej książki wśród milionów jako główne czynniki tego procesu. Z myślą o tym problemie powstał pomysł na LitMatch - prototyp chatbota, który ma na celu zmniejszyć czas potrzebny na znalezienie nowych utworów do przeczytania oraz zrobić ten proces bardziej spersonalizowanym i dostępnym.

Celem pracy jest utworzenie funkcjonalnego prototypu chatbota, który używa generowania wspomaganego pobieraniem do rekomendacji książek na podstawie preferencji użytkownika. Prototyp działa zarówno w języku polskim, jak i w języku angielskim, oraz korzysta z wzbogaconego przez wielki model językowy zestawu danych, który zawiera informacje o 843 utworach.

Praca składa się z czterech rozdziałów. **W pierwszym rozdziale** zostanie omówiony problem zanikającego interesu ludzi do czytania oraz w jaki sposób chatbot może rozwiązać ten problem. Pomimo tego, rozdział przedstawi zakres pracy.

Rozdział drugi zostanie poświęcony omówieniu zagadnień, znajomość których jest ważna do zrozumienia niniejszej pracy. Do wspomnianych zagadnień należą: osadzenia wektorowe, wielkie modele językowe oraz generowanie wspomaganie pobieraniem.

W trzecim rozdziale zostaną przedstawione szczegóły implementacji prototypu, od utworzenia bazy danych do utworzenia interfejsu graficznego chatbota.

¹ Jeffrey M. Jones, *Americans Reading Fewer Books Than in Past* [online],

<https://news.gallup.com/poll/388541/americans-reading-fewer-books-past.aspx> [dostęp: 17.05.2026]

² Jessica K. Bone, Feifei Bu, Jill K. Sonke, Daisy Fancourt, *The decline in reading for pleasure over 20 years of the American Time Use Survey*, CellPress, "iScience" z 19.09.2025, nr. 9

Ostatni rozdział przedstawia testy, które zostały wykonane w celu sprawdzenia poprawności działania chatbota.

ROZDZIAŁ 1. MOTYWACJA I ZAKRES PRACY

1.1 Chatboty

Chatbot to programem komputerowym, który jest zaprojektowany w sposób, aby symulować ludzkie konwersacje. Pierwszy chatbot, ELIZA, który powstał w 1966 roku, imitował psychiatrę. ELIZA korzystała z rozpoznawania wzorców i wyboru odpowiedzi do komunikacji z użytkownikami. Pomimo tego, wiedza wspomnianego chatbota była ograniczona do jednej dziedziny. Nie mniej jednak ELIZA była inspiracją dla projektów, które powstawały w kolejnych latach. Jedną z wcześniejszych prób integracji sztucznej inteligencji z chatbotem był Jabberwacky w 1988 roku, wtedy jak bardziej zdolny system o nazwie ALICE powstał w 1995 roku. W przeciwieństwie do ELIZA, chatbot ALICE posiadał wiedzę z różnych dziedzin, jednak nie potrafił generować odpowiedzi podobnych do ludzkich, które mogłyby wyrażać emocje czy nastawienie. W 2001 roku powstał bot SmarterChild. Dostępny w messengerach America Online i Microsoft, SmarterChild pozwalał swoim użytkownikom na pobieranie z baz danych informacji, takich jak czas początku seansu w kinie, ceny papierów wartościowych, wiadomości i prognozy pogody. Kolejnym etapem ewolucji było pojawienie się asystentów głosowych Siri, IBM Watson, Google Assistant, Amazon Alexa itd. Wszystkie asystenci dobrze radzili sobie z monitorowaniem kalendarzy, innych urządzeń czy poczty, ale miały różne problemy i przede wszystkim nie rozumieli kontekstu całej konwersacji.

Do prawdziwego przełomu doszło 30 listopada 2022 roku, kiedy to firma OpenAI udostępniła ChatGPT szerokiej publiczności. W przeciwieństwie do swoich poprzedników, ChatGPT może się wypowiadać na dowolny temat, korzysta uczenia pogłębionego i rozumie kontekst konwersacji, którą z nim prowadzi użytkownik. Był to także pierwszy powszechnie dostępny chatbot, który dawał odpowiedzi bardzo podobne do wypowiedzi ludzkich. Z upływem czasu zaczęły powstawać inne systemy o wysokich zdolnościach, na przykład Claude firmy Anthropic i Copilot firmy Microsoft. Ta ewolucja całkowicie zmieniła to, w jaki sposób ludzie porozumiewali się z komputerami. Nowoczesne chatboty pozwalają użytkownikom wyrażać swoje potrzeby własnymi słowami i nie wymagają znajomości terminologii technicznej czy akademickiej, ponieważ wielkie modele językowe rozumieją język codzienny tak samo dobrze jak i język formalny.

1.2 Czytelnictwo w XXI wieku

W 2025 roku zostały opublikowane wyniki badania śledzącego popularność czytania dla przyjemności oraz czytania z dziećmi wśród Amerykanów z 2003 po 2023 rok. Badanie zostało przeprowadzone przez University of Florida wspólnie z University College London na próbie liczącej 237 270 osób. Wyniki sugerują, że w badanym okresie liczba ludzi codziennie czytających w Stanach Zjednoczonych Ameryki spadała rocznie o 3%. W 2023 roku, tylko 1747 osób czytało codziennie dla przyjemności. Ponadto, autorzy artykułu twierdzą, że tendencja spadku była notowana od lat 40. ubiegłego wieku. Przyczyną tego negatywnego zjawiska jest zmniejszająca się liczba czasu wolnego, problemy z koncentracją oraz media społecznościowe i elektroniczne, które nie wymagają od człowieka żadnego wysiłku. Niewystarczająca ilość czasu komplikuje także proces poszukiwania książki do przeczytania, gdyż człowiek ma albo wybrać się do biblioteki po ośmiu godzinach pracy albo poświęcić kilka minut, aby zapoznać się z postami i artykułami w Internecie, na podstawie których mógłby dokonać wyboru. Chatbot LitMatch stanowi próbę częściowego rozwiązania tego problemu poprzez uproszczenie procesu wyboru kolejnej lektury. System rekomenduje książki w ciągu kilku sekund, nie wymagając od użytkownika specjalistycznej wiedzy literackiej — wystarczy opisać swoje preferencje własnymi słowami lub podać tytuł książki, którą się podobała. W przeciwieństwie do list bestsellerów czy ogólnych rankingów, LitMatch oferuje rekomendacje spersonalizowane, dopasowane do indywidualnych upodobań czytelnika.

1.3 Motywacja

Dana praca powstała jako próba rozwiązania własnego problemu ze znalezieniem i wyborem nowych książek do przeczytania. Studiując na Litwie, korzystanie z lokalnych bibliotek jest utrudnione ze względu na barierę językową — nawet jeśli otrzymałbym rekomendację, większość dostępnych pozycji byłaby niedostępna w języku polskim lub angielskim. Wolę otrzymywać rekomendacje poprzez rozmowę, a nie przez przeglądanie list bestsellerów czy artykułów w Internecie, które rzadko uwzględniają indywidualne preferencje czytelnika. Liczba dostępnych książek jest ogromna i samodzielne śledzenie wszystkich utworów wartych uwagi jest niemożliwe. Ponieważ interesuję się również sztuczną inteligencją, wpadłem na pomysł stworzenia LitMatch — chatbota, który pozwalałby otrzymywać spersonalizowane rekomendacje w języku naturalnym, poprzez interakcję

przypominającą rzeczywistą rozmowę z człowiekiem. W przeciwieństwie do bibliotekarza obsługującego wielu czytelników jednocześnie, LitMatch skupia się wyłącznie na potrzebach jednego użytkownika, co pozwala na głębszą personalizację rekomendacji.

1.4 Zakres pracy

Zakres pracy nad danym projektem obejmował:

- utworzenie bazy danych PostgreSQL
- ładowanie danych z pliku .csv do bazy danych
- utworzenie wektorowej bazy Chroma
- wzbogacenie zawartości bazy danych PostgreSQL za pomocą wielkiego modelu językowego
- przekształcenie wzbogaconych informacji o książkach na wektory i zapis do bazy wektorowej Chroma
- utworzenie chatbota polecającego książki na podstawie generowania wspomaganego pobieraniem

ROZDZIAŁ 2. KLUCZOWE POJĘCIA I TECHNOLOGIE

2.1 Osadzenia wektorowe

Ponieważ modele uczenia maszynowego, takie jak uczenie głębokie i regresja liniowa, w swojej pracy korzystają z logiki matematycznej, ich dane wejściowe, niezależnie od typu, mają być przedstawione w postaci liczbowej. Transformacji danych na postać numeryczną dokonuje się za pomocą osadzeń wektorowych. Osadzenia wektorowe (*ang.* vector embeddings) są techniką, która pozwala zamienić złożone, pozbawione struktury dane w uporządkowane wektory liczbowe. Każdy z powstałych wektorów jest odwzorowaniem oryginalnego obiektu, który zachowuje jego kluczowe cechy. To pozwala modelom uczyć się rozpoznawać prawdopodobieństwa i różnice między obiektami, i.e. podobne dane są reprezentowane przez wektory o zbliżonych wartościach. W kontekście niniejszej pracy osadzeń wektorowych użyto do wyboru książek, cechy których najbardziej odpowiadają preferencjom użytkownika.

2.2 Wielkie modele językowe

Kolejnym istotnym pojęciem są wielkie modele językowe. Są to zaawansowane systemy sztucznej inteligencji, które są zaprojektowane do przetwarzania i generowania języka naturalnego. Podstawą wielkich modeli językowych zwykle jest architektura transformera, która pozwala na analizowanie długich tekstów i wychwytywanie złożonych relacji językowych. Ponieważ wspomniane wcześniej modele są trenowane na wielkich zestawach danych, posiadają wiedzę z różnych dziedzin nauki i życia ludzkiego. W swoich odpowiedziach wielki model językowy przewiduje kolejne słowa na podstawie prawdopodobieństw i wcześniejszego kontekstu. W niniejszej pracy wielkiego modelu językowego Llama 4 Scout użyto do wzbogacenia informacji o książkach oraz tworzenia rekomendacji w postaci języka naturalnego.

2.3 Generowanie wspomagane pobieraniem

Najistotniejszym pojęciem w kontekście danego projektu jest generowanie wspomagane pobieraniem (*ang.* retrieval-augmented generation, RAG). Jest to proces, który polega na optymalizacji wyników działania wielkiego modelu językowego, aby przed generowaniem odpowiedzi odnosił się do wiarygodnego źródła wiedzy znajdującego poza zestawem danych,

na którym był trenowany. Bez RAG, wielkie modele językowe korzystają z zapytania użytkownika i generują odpowiedź na podstawie istniejącej wiedzy. Z RAG zapytanie użytkownika zostanie zamienione w wektor, który zostanie porównany do zewnętrznego źródła. Na podstawie porównania wektorów pobierane są aktualne dane. Treść wprowadzona przez użytkownika zostanie później uzupełniona o informacje pobrane z zewnątrz i przekazana do wielkiego modelu językowego. Z procesu generowania uzupełnionego pobieraniem będzie korzystał opisywany w niniejszej pracy prototyp chatbota.

ROZDZIAŁ 3. REALIZACJA

3.1 Plik .env

Pierwszym plikiem, który został utworzony w ramach pracy nad projektem, jest plik **.env**. Zawiera on poniższe zmienne:

- GROQ_API_KEY – klucz API, który pozwala łączyć się z wielkim modelem językowym
- DB_HOST – nazwa hosta, która jest potrzebna do połączenia się z bazą danych przechowującą tabelę z informacją o książkach
- DB_PORT – port, przez który nawiązuje się połączenie z bazą danych przechowującą tabelę z informacją o książkach
- DB_NAME – nazwa bazy danych, która przechowuje informację o książkach
- DB_USER – nazwa użytkownika, która jest wykorzystywana podczas nawiązania połączenia z bazą danych
- DB_PASSWORD – hasło do bazy danych przechowującej tabelę z informacją o książkach

Ze względów bezpieczeństwa, treść pliku nie zostanie przedstawiona w pracy, ponieważ zawiera wrażliwe informacje.

3.2 Plik create_table.py

Kolejnym plikiem będącym częścią projektu jest `create_table.py`. Plik ten zawiera kod tworzący tabelę, do której zostaną zapisane informacje o książkach z pliku Excel. Treść pliku `create_table.py` przedstawia poniższy obraz:

Rysunek 1. Plik `create_table.py`

```
import os
import psycopg2
from dotenv import load_dotenv

load_dotenv()
```

```

host = os.getenv("DB_HOST")
port = os.getenv("DB_PORT")
db = os.getenv("DB_NAME")
user = os.getenv("DB_USER")
pwd = os.getenv("DB_PASSWORD")

sql = """
CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(500),
    author VARCHAR(300),
    description TEXT,
    genres TEXT[],
    average_rating NUMERIC(3,2),
    number_of_ratings INTEGER,
    themes TEXT[],
    mood VARCHAR(100),
    pacing VARCHAR(50),
    literary_period VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW()
);"""

conninfo = f"host={host} port={port} dbname={db} user={user} password={pwd}"

with psycopg2.connect(conninfo) as conn:
    with conn.cursor() as cur:
        cur.execute(sql)

```

Źródło: kod własny

Jak wynika z Rysunku 1, kod zaczyna się od importu modułów `os`, `psycopg2` oraz funkcji `load_dotenv`. Moduł `os` służy do pracy z systemem operacyjnym, `psycopg2` pozwala na łączenie się z bazą danych typu PostgreSQL przez kod, a `load_dotenv` odczytuje pary kluczy i wartości z pliku środowiska. Odczytane wartości później zostają przypisane zmiennym. Zmiennej `host` zostaje przypisana wartość klucza `DB_HOST`, zmiennej `port` – wartość `DB_PORT`. Zmienne `db`, `user` i `pwd` będą przechowywały odpowiednio wartości kluczy `DB_NAME`, `DB_USER` i `DB_PASSWORD`.

Kolejnym elementem skryptu jest zmienna `sql` przechowująca zapytanie SQL, które ma utworzyć tabelę, gdzie będą przechowywane informacje o książkach. Tabela o nazwie `books` będzie posiadała poniższe kolumny:

- `id` – numer, który będzie pełnił rolę klucza podstawowego
- `title` – nazwa książki w postaci ciągu o maksymalnej długości 500 znaków
- `author` – imię i nazwisko autora w postaci ciągu o maksymalnej długości 300 znaków
- `description` – opis książki w postaci ciągu znaków
- `genres` – gatunki książki przedstawione w postaci listy
- `average_rating` – średnia ocena książki na stronie `goodreads.com`, przedstawiona w postaci liczby z dwoma znakami po przecinku
- `numer_of_ratings` – liczba opinii o utworze
- `themes` – wątki literackie przedstawione w postaci listy
- `mood` – nastrój utworu w postaci ciągu znaków o maksymalnej długości 100 znaków
- `pacing` – tempo narracji w postaci ciągu znaków o maksymalnej długości 50 znaków
- `literary_period` – epoka literacka utworu w postaci ciągu o maksymalnej długości 100 znaków
- `created_at` – znacznik czasu pokazujące kiedy rekord został dodany do tabeli

Po zmiennej `sql` powstaje zmienna `conninfo`. Zawiera ciąg składający się z informacji potrzebnych do nawiązania połączenia z bazą danych, czyli z nazwy hosta, numeru portu, nazwy bazy danych, nazwy użytkownika i hasła. Informacje dla `conninfo` są pobierane ze zmiennych `host`, `port`, `db`, `user` i `pwd` wspomnianych powyżej. Skrypt kończy się nawiązaniem połączenia z bazą danych i wykonaniem zapytania zawartego w zmiennej `sql`.

3.3 load_data.py

Jako trzeci w ramach projektu powstał plik `load_data.py`, który przedstawia Rysunek 2:

Rysunek 2. Plik `load_data.py`

```
import pandas as pd
from sqlalchemy import create_engine
import ast
from dotenv import load_dotenv
import os
```

```
load_dotenv()

host = os.getenv("DB_HOST")
port = os.getenv("DB_PORT")
db = os.getenv("DB_NAME")
user = os.getenv("DB_USER")
pwd = os.getenv("DB_PASSWORD")
file_path = "C:/Users/m.novikov/diploma_project/goodreads_data.csv"

raw_file = pd.read_csv(file_path)
file_df = pd.DataFrame(raw_file)
engine = create_engine(f"postgresql://{user}:{pwd}@{host}:{port}/{db}")

file_df = file_df.drop(columns=["Unnamed: 0", "URL"])

file_df = file_df.dropna(subset=["Description"])

file_df =
file_df.rename(columns={"Book": "title", "Author": "author", "Description": "description", "Genres": "genres",
"Avg_Rating": "average_rating", "Num_Ratings": "number_of_ratings"})

file_df["genres"] = file_df["genres"].apply(ast.literal_eval)

file_df["number_of_ratings"] = file_df["number_of_ratings"].str.replace(",", "", "").astype(int)

file_df.to_sql("books", con=engine, index=False, if_exists="append")
```

Źródło: kod własny

Zarówno jak `create_table.py`, `load_data.py` zaczyna się od importu modułów i funkcji. Pierwszy z nich, `pandas`, jest później wykorzystywany do odczytania danych o książkach z pliku Excel oraz ich transformacji. Później z modułu `sqlalchemy` importuje się funkcję `create_engine`, która będzie potrzebna do nawiązania połączenia z bazą danych. Kolejny moduł, `ast`, będzie służył do przekształcenia tekstowych reprezentacji literału na rzeczywisty obiekt w języku Python. Funkcja `load_dotenv` z modułu `dotenv`, podobnie jak w pliku `create_table.py`, jest używana do odczytania kluczy i wartości z pliku środowiska. Ostatni z zaimportowanych modułów, `os`, wykorzystuje się do pracy z systemem operacyjnym.

Po imporcie modułów i funkcji tworzone są zmienne `host`, `port`, `db`, `user` i `pwd`, które będą przechowywały odpowiednio nazwę hosta, numer portu, nazwę bazy danych, jej użytkownika i hasło. Wartości zmiennych odczytuje się z pliku `.env` za pomocą funkcji `load_dotenv`. Kolejna zmienna, `file_path`, przechowuje drogę do pliku `goodreads_data.csv` przechowującego informacje o książkach, wtedy jak w engine znajduje się ciąg znaków z parametrami niezbędnymi do nawiązania połączenia z bazą danych. Treść pliku Excel zostaje później odczytana za pomocą modułu `pandas` i przypisana do zmiennej `file_df` w postaci `DataFrame`. `DataFrame` poddaje się transformacji: zostają usunięte kolumny `Unnamed: 0` i `URL` oraz wszystkie rekordy, w których kolumna `Description` jest pusta. W nazwach pozostałych kolumn wielkie litery są zamieniane na małe. Wartości z kolumny `genres` (ang. gatunki) przemienia się na obiekty Python, a z wartości w kolumnie `number_of_ratings` usuwa się przecinki. Po transformacji dane są zapisywane do tabeli `books` za pomocą metody `to_sql` w trybie `append`, co oznacza, że nowe informacje o książkach są dopisywane do tabeli i operacja dodawania danych nie wpływa na wcześniej dodane rekordy.

3.4 enrich_data.py

Kolejnym plikiem projektu jest `enrich_data.py`. Kod zawarty w tym pliku służy do wzbogacania danych z tabeli `books` o wątki literackie, nastrój, tempo narracji i epokę literacką.

Zarówno jak poprzednie pliki, `enrich_data.py` zaczyna się od importu modułów i funkcji:

Rysunek 3. Import modułów i funkcji w pliku `enrich_data.py`

```
import os
from dotenv import load_dotenv
from groq import Groq
import psycopg2
import json

load_dotenv()
```

```
host = os.getenv("DB_HOST")
port = os.getenv("DB_PORT")
db = os.getenv("DB_NAME")
user = os.getenv("DB_USER")
pwd = os.getenv("DB_PASSWORD")
api_key = os.getenv("GROQ_API_KEY")
```

Źródło: kod własny

Pierwszym zaimportowanym modulem jest `os`. Służy on do pracy z systemem operacyjnym. Zadaniem funkcji `load_dotenv`, jak i wcześniej, jest ładowanie parametrów potrzebnych do nawiązania połączenia z bazą danych przechowującą informacje o książkach oraz klucz API przechowywany w pliku `.env` przez `GROQ_API_KEY`. Klucz API zostaje przypisany do zmiennej `api_key`, wtedy jak pozostałe parametry, czyli numer portu, nazwa bazy danych, nazwa użytkownika i hasło zostają przypisane odpowiednio do zmiennych `port`, `db`, `user` i `pwd`. Później importuje się klasę `Groq` z biblioteki `groq`. Klasa zostanie wykorzystana do komunikacji z Groq API. Kolejny zaimportowany moduł, `psycopg2`, używa się, jak i w plikach opisanych wcześniej, do nawiązania połączenia z bazą danych przechowującą książki. Jako ostatni importuje się moduł `json`, który zostaje wykorzystany do pracy z danymi zwróconymi przez model językowy.

Ważnym elementem skryptu jest zapytanie, które przedstawia poniższy rysunek:

Rysunek 4. Zapytanie w pliku `enrich_data.py`

```
prompt = """
You are an experienced literature professor with decades of expertise in world
literature
and literary history. Your task is to analyze books and return structured metadata
in JSON format.

You will be given a book's title, author, and description. Use your own knowledge
of the book
if you are familiar with it. If you are not familiar with the book, infer the
attributes
from the description provided.

For each book, return ONLY a valid JSON object with exactly these fields:
```

```
{
  "themes": [list of up to 5 themes, e.g. "redemption", "survival", "forbidden
love", "coming of age", "revenge"],
  "mood": one of: "dark", "mysterious", "romantic", "humorous", "tense",
"uplifting", "melancholic", "suspenseful", "adventurous", "philosophical",
  "pacing": one of: "slow", "moderate", "fast",
  "literary_period": one of: "Antiquity", "Middle Ages", "Renaissance",
"Baroque", "Enlightenment", "Romanticism", "Realism", "Modernism", "Contemporary"
}

Rules:
- Return ONLY the JSON object, no explanation, no additional text
- If a field cannot be determined confidently, return null for that field
- Themes should be concise, 1-3 words each
- Mood must be exactly one value from the provided list
- Pacing must be exactly one value from the provided list
- Literary period must be exactly one value from the provided list
"""
```

Źródło: kod własny

Zapytanie opisuje w jaki sposób ma się zachowywać wielki model językowy. W danym przypadku, model ma się wcielić w doświadczonego profesora literatury z wielkim doświadczeniem. Zadaniem modelu jest analiza podanych książek i zwrócenie metadanych w formacie JSON. Informacje, których się oczekuje od modelu, to wątki literackie, nastrój, tempo narracji i epoka literacka. Model ma pobrać informacje z własnej wiedzy lub wyciągnąć je tego, co zostało podane na wejściu. Zapytanie wprowadza ograniczenia na wartości, które poszczególne atrybuty mogą przyjmować. Może podać maksymalnie pięć wątków literackich dla jednego utworu, jeden z dziesięciu nastrojów, jedno tempo narracji i jedną ze znanych epok literackich. Zapytanie kończy się zasadami, które wymagają, aby model zwracał tylko obiekt typu JSON bez jakichkolwiek wyjaśnień, zwracał null dla pól, wartości których nie udało się wyznaczyć i podkreśla, że nastrój, tempo i epoka literacka muszą mieć jedną wartość z listy dopuszczalnych wartości.

Po prompcie tworzy się zmienne i z bazy danych są pobierane informacje o książkach, co przedstawia Rysunek 5:

Rysunek 5. Połączenie z bazą danych i pobranie informacji o książkach w pliku `enrich_data.py`

```
conninfo = f"host={host} port={port} dbname={db} user={user} password={pwd}"

SQL = "SELECT id, title, author, description FROM books WHERE mood is NULL OR
themes is NULL OR pacing is NULL OR literary_period is NULL"

with psycopg2.connect(conninfo) as conn:
    with conn.cursor() as cur:
        cur.execute(SQL)
        rows = cur.fetchall()

helping_fields = {}

for row in rows:
    helping_fields[row[0]] = {
        "title": row[1],
        "author": row[2],
        "description": row[3]
    }

client = Groq(
    api_key=api_key
)
```

Źródło: kod własny

Zmiennej `conninfo` zostaje przypisany ciąg znaków potrzebny do połączenia z bazą danych, a zmienna `SQL` będzie przechowywała zapytanie do bazy danych, które ma pobrać wszystkie rekordy, które nie mają nastroju, wątków literackich, tempa narracji bądź epoki literackiej.

Później za pomocą modułu `psycopg2` oraz zmiennej `conninfo` na bazie danych zostaje wykonane zapytanie zawarte w zmiennej `SQL`. Wynik zapytania zostaje zapisany do zmiennej `rows`. Z `rows` atrybuty wspomagające, czyli nazwa utworu, jego autor i opis zostaną zapisane do zagnieżdżonego słownika `helping_fields`. Po zapisie do słownika utworzona zostaje zmienna `client`, która będzie przechowywała połączenie z Groq API.

Najważniejszą częścią kodu zawartego w pliku `enrich_data.py` jest mechanizm wzbogacenia danych przedstawiony na poniższym rysunku:

Rysunek 6. Mechanizm wzbogacenia informacji o książkach z pliku `enrich_data.py`

```
with psycopg2.connect(conninfo) as conn:
    with conn.cursor() as cur:
        for book_id, record in helping_fields.items():
            title = record["title"]
            author = record["author"]
            description = record["description"]

            chat_completion = client.chat.completions.create(
                messages=[
                    {"role": "system", "content": prompt},
                    {"role": "user", "content": f"Title: {title}\nAuthor: {author}\nDescription: {description}"}
                ],
                model="meta-llama/llama-4-scout-17b-16e-instruct"
            )

            response = chat_completion.choices[0].message.content
            response = response.strip().removeprefix("`json").removeprefix("`").removesuffix("`").strip()

            try:
                data = json.loads(response)
                themes = data["themes"]
                mood = data["mood"]
                pacing = data["pacing"]
                literary_period = data["literary_period"]

                cur.execute("""
                    UPDATE books
                    SET themes = %s, mood = %s, pacing = %s,
literary_period = %s
                    WHERE id = %s
                """, (themes, mood, pacing, literary_period, book_id))
                conn.commit()
```

```
except:
    print(f"Failed to parse response for book {book_id}, skipping")
    continue
```

Źródło: kod własny

Wzbogacenie informacji zaczyna się otwarciem połączenia z bazą danych PostgreSQL. Później dochodzi do pobrania ze słownika `helping_fields` nazwy utworu, jego autora i opisu. W kolejnym kroku tworzy się zmienną `chat_completion`. Zawiera ona informacje, które zostaną przekazane Groq API, czyli rolę, którą ma odgrywać wielki model językowy, dane wejściowe (nazwa utworu, autor, opis) oraz nazwa modelu, czyli Llama 4 Scout. Odpowiedź wielkiego modelu językowego zostaje zapisana w zmiennej `response`. Z odpowiedzi usuwa się niepożądane znaki i zostaje podjęta próba załadowania informacji do zmiennej `data`. Z `data` wyciąga się wątki literackie, nastrój, tempo narracji i epokę literacką utworu, które zostają zapisane odpowiednio w zmiennych `themes`, `mood`, `pacing` i `literary_period`. Treść zmiennych później jest wklejana do zapytania SQL, które ma zmodyfikować rekord w bazie danych i dodać informacje otrzymane od wielkiego modelu językowego. W przypadku niepowodzenia, kod zwróci powiadomienie o błędzie.

3.5 book_to_vector.py

Jednym z ostatnich plików utworzonych w ramach projektu jest `book_to_vector.py`. Plik zaczyna się od importu modułów, co przedstawia poniższy rysunek:

Rysunek 7. Import modułów i funkcji w pliku `book_to_vector.py`

```
import os
import psycopg2
from dotenv import load_dotenv
import re
import chromadb

load_dotenv()

host = os.getenv("DB_HOST")
port = os.getenv("DB_PORT")
db = os.getenv("DB_NAME")
```

```
user = os.getenv("DB_USER")
pwd = os.getenv("DB_PASSWORD")
api_key = os.getenv("GROQ_API_KEY")
```

Źródło: kod własny

Jako pierwszy importuje się moduł `os` do pracy z systemem operacyjnym urządzenia, z którego zostaje uruchomiony kod projektu. Moduł `psycopg2` jak i wcześniej służy do nawiązania połączenia z bazą danych PostgreSQL, a funkcja `load_dotenv` ma załadować ze środowiska wartości parametrów. Wartości te później zostają przypisane zmiennym `host`, `port`, `db`, `user`, `pwd` i `api_key`. Moduł `re` zostanie wykorzystany do manipulacji nad ciągami znaków, wtedy jak `chromadb` posłuży do utworzenia bazy danych, która będzie przechowywała informacje o książkach w postaci wektorów.

Kolejnym etapem jest połączenie się z bazą danych PostgreSQL i pobranie stąd informacji o książkach. Ten etap przedstawia poniższy rysunek:

Rysunek 8. Nawiązanie połączenia z bazą danych i pobranie informacji w pliku `book_to_vector.py`

```
conninfo = f"host={host} port={port} dbname={db} user={user} password={pwd}"

SQL = """
SELECT
    id,
    title,
    author,
    genres,
    themes,
    mood,
    pacing,
    literary_period,
    average_rating
FROM books
"""

def remove_symbols(text):
    cleaned = re.sub(r'^A-Za-z0-9\s,','', text)

    return cleaned
```

```
with psycopg2.connect(conninfo) as conn:
    with conn.cursor() as cur:
        cur.execute(SQL)
        rows = cur.fetchall()

data_dict = {}

for row in rows:
    data_dict[row[0]] = {
        "title": row[1],
        "author": row[2],
        "genres": row[3],
        "themes": row[4],
        "mood": row[5],
        "pacing": row[6],
        "literary_period": row[7],
        "average_rating": row[8]
    }
```

Źródło: kod własny

Po pobraniu wartości parametrów ze środowiska powstaje zmienna `conninfo`, która łączy wszystkie parametry w jednym ciągu. Kolejna zmienna, `SQL`, przechowuje zapytanie, które ma pobrać całą zawartość tabeli `books`. Później zostaje zdefiniowana funkcja `remove_symbols`, która zostanie później użyta do usuwania niepożądanych symboli. W kolejnym kroku za pomocą modułu `psycopg2` nawiązuje się połączenie z bazą danych przechowującą informacje o książkach. Na bazie danych wykonuje się zapytanie przypisane do zmiennej `SQL`. Z wyniku zapytanie zostają pobrane: nazwa książki, jej autor, gatunki, wątki literackie, nastrój, tempo narracji, epoka literacka i średnia ocena. Wszystkie te wartości zostają zapisane do zagnieżdżonego słownika `data_dict`.

Kolejnym krokiem jest przemiana treści przechowywanych przez `data_dict` na format przyjmowany przez ChromaDB. Ten proces przedstawia Rysunek 9:

Rysunek 9. Normalizacja treści `data_dict`

```
book_ids = []
books_strings = []
books_metadata = []
```

```
for id, record in data_dict.items():
    genres_string = remove_symbols(", ".join(record["genres"]))
    themes_string = remove_symbols(", ".join(record["themes"]))

    book_string = f"""Title: {record["title"]}. Author: {record["author"]}.
Genres: {genres_string}. Themes: {themes_string}.
Mood: {record["mood"]}. Pacing: {record["pacing"]}. Literary period:
{record["literary_period"]}
"""

    book_metadata_dict = {}
    book_metadata_dict["title"] = record["title"]
    book_metadata_dict["author"] = record["author"]
    book_metadata_dict["genres"] = genres_string
    book_metadata_dict["themes"] = themes_string
    book_metadata_dict["mood"] = record["mood"] or ""
    book_metadata_dict["pacing"] = record["pacing"] or ""
    book_metadata_dict["literary_period"] = record["literary_period"] or ""
    book_metadata_dict["average_rating"] = float(record["average_rating"])

    book_ids.append(str(id))
    books_strings.append(book_string)
    books_metadata.append(book_metadata_dict)
```

Źródło: kod własny

Transformacja zaczyna się od usunięcia przecinków z kolumn reprezentujących gatunki i wątki utworu. Później powstaje zmienna `book_string`, która przechowuje dane o książce w formacie dokumentu. Do słownika `metadata_dict` zostają zapisane wszystkie atrybuty utworu literackiego, tworząc w taki sposób metadane. Jeżeli dla książki brakuje nastroju, tempa narracji lub epoki literackiej, brakującemu atrybutu zostanie przypisana wartość `None`. Dla kolumny reprezentującej średnią ocenę wartość z `data_dict` zostaje przekonwertowana na format `float`. Później id książki w postaci ciągu znaków zostaje dopisany do listy `book_ids`, zmienna `book_string` do listy `books_strings`, a słownik `books_metadata_dict` – do listy `books_metadata`.

Plik kończy się na utworzeniu wektorowej bazy danych dla książek.

Rysunek 10. Utworzenie wektorowej bazy danych

```
chroma_client =
chromadb.PersistentClient(path="C:/Users/m.novikov/diploma_project/chroma_db")

collection = chroma_client.create_collection(name="books")
collection.add(
    ids=book_ids,
    documents=books_strings,
    metadatas=books_metadata
)
```

Źródło: kod własny

Jak wynika z Rysunku 10, do zmiennej `chroma_client` przypisuje się ścieżkę do bazy danych `chroma_db`. Następnie tworzy się kolekcję o nazwie `books`, która będzie przechowywana przez zmienną `collection`. Ostatnim krokiem jest dodanie do kolekcji list `book_ids`, `books_strings` i `books_metadata`.

3.6 query_chatbot.py

Ostatnim plikiem projektu jest `query_chatbot.py`. Jak i wszystkie pliki projektu opisane wcześniej, zaczyna się od importu modułów i funkcji:

Rysunek 11. Import modułów i funkcji w pliku `query_chatbot.py`

```
from groq import Groq

from dotenv import load_dotenv

import os

import chromadb

import streamlit as st

import json

import base64
```

```
load_dotenv()

api_key = os.getenv("GROQ_API_KEY")

client = Groq(api_key=api_key)
```

Źródło: kod własny

Jako pierwsza zostaje zaimportowana klasa Groq, która zostanie wykorzystana do nawiązania połączenia z modelem językowym. Funkcji load_dotenv używa się do załadowania par kluczy i wartości ze środowiska, a modułu os – do przypisania zmiennej api_key wartości klucza GROQ_API_KEY. Kolejny moduł, chromadb, zostanie użyty do połączenia z wektorową bazą danych, wtedy jak za pomocą streamlit zostanie stworzony interfejs graficzny. Następny moduł, json, będzie służył do pracy z odpowiedziami wielkiego modelu językowego. Ostatni z zaimportowanych modułów, base64, będzie służył do przemiany zdjęcia pieczęci w ciąg znaków.

Istotnym elementem skryptu jest słownik LANGUAGES, który przedstawia poniższy rysunek:

Rysunek 12. Słownik LANGUAGES w pliku query_chatbot.py

```
LANGUAGES = {

    "English": {

        "extraction_prompt": ""

        You are a query understanding assistant for a book recommendation system.

        Analyze the user's message and extract book preference parameters.

        Return ONLY a valid JSON object with exactly these fields:

        {
```

```
"mood": one of: "dark", "mysterious", "romantic", "humorous", "tense",
"uplifting", "melancholic", "suspenseful", "adventurous", "philosophical" or null,

    "genres": list of relevant genres from: ["Fantasy", "Science Fiction",
"Romance", "Horror", "Crime", "Thriller", "Non-fiction", "Historical Fiction",
"Classics", "Self Help"] or null,

"themes": list of up to 3 themes mentioned or inferred, or null,

    "pacing": one of: "slow", "moderate", "fast" or null,

    "literary_period": one of: "Antiquity", "Middle Ages", "Renaissance",
"Baroque", "Enlightenment", "Romanticism", "Realism", "Modernism", "Contemporary"
or null,

    "reference_book": title of any specific book the user mentions or null,

    "author": name of any specific author the user mentions or null,

    "vague": true if the query is too general to extract any meaningful
parameters, false otherwise
}
```

Rules:

- Return ONLY the JSON object, no explanation, no additional text
- Set "vague" to true only if you cannot extract any useful parameters at all
- If the user mentions a specific book, always capture it in "reference_book"
- If the user mentions a specific author, always capture it in "author"
- If the user mentions an author, set "vague" to false

```
""
```

```
    "recommendation_prompt": ""
```

You are a friendly and enthusiastic librarian who loves helping people find their next great read. Your job is to recommend books based on the user's preferences and the book options you are given as context.

Keep your responses warm, informal, and concise – like a friend giving advice,

not a professor giving a lecture. For each recommendation, give a short and engaging explanation of why the user would enjoy it.

You MUST only recommend books that are explicitly listed in the context provided to you. Never recommend books from your own knowledge. If the context doesn't contain suitable books, say so honestly and ask the user to refine their request.

If the user asked about a specific author and no books by that author appear in the context,

say clearly that you don't have any books by that author in the database, and offer to

recommend something similar instead. Do NOT suggest specific book titles from your own knowledge.

Never show your reasoning or thinking process. Never use phrases like "Let me think",

"Based on the context", "I can see that", or any other meta-commentary.

Just respond naturally and directly.

Always respond in English. Never use any other language.

""",

"about_text": "This chatbot recommends books based on your preferences using AI and a curated database of 843 books.",

"new_conversation": "New Conversation",

"spinner_text": "Finding books for you...",

"input_placeholder": "Tell me what you're in the mood for...",

"not_found_message": "I'm sorry, it looks like **{book}** isn't in my database. Would you like me to recommend something based on your other preferences, or could you describe what you enjoyed about that book?",

```
"error_search": "Something went wrong with the search. Please try again.",
    "error_response": "Something went wrong. Please try again.",
},

"Polski": {
    "extraction_prompt": ""

Jesteś asystentem rozumienia zapytań dla systemu rekomendacji książek.

Przeanalizuj wiadomość użytkownika i wyodrębni parametry preferencji dotyczących
książek.

Zwróć TYLKO prawidłowy obiekt JSON z dokładnie tymi polami:

{
    "mood": jedno z: "dark", "mysterious", "romantic", "humorous", "tense",
"uplifting", "melancholic", "suspenseful", "adventurous", "philosophical" lub
null,

    "genres": lista odpowiednich gatunków z: ["Fantasy", "Science Fiction",
"Romance", "Horror", "Crime", "Thriller", "Non-fiction", "Historical Fiction",
"Classics", "Self Help"] lub null,

    "themes": lista do 3 tematów wspomnianych lub wywnioskowanych, lub null,

    "pacing": jedno z: "slow", "moderate", "fast" lub null,

    "literary_period": jedno z: "Antiquity", "Middle Ages", "Renaissance",
"Baroque", "Enlightenment", "Romanticism", "Realism", "Modernism", "Contemporary"
lub null,

    "reference_book": tytuł konkretnej książki wspomnianej przez użytkownika lub
null,

    "author": imię i nazwisko konkretnego autora wspomnianego przez użytkownika
lub null,

    "vague": true jeśli zapytanie jest zbyt ogólne, aby wyodrębnić jakiegokolwiek
znaczące parametry, false w przeciwnym razie

}
```

Zasady:

- Zwróć TYLKO obiekt JSON, bez wyjaśnień, bez dodatkowego tekstu
- Ustaw "vague" na true jeśli użytkownik nie podaje żadnych konkretnych preferencji dotyczących gatunku, nastroju, tematu, tempa, autora lub konkretnej książki
- Jeśli użytkownik wspomina konkretną książkę, zawsze zapisz ją w "reference_book"
- Jeśli użytkownik wspomina konkretnego autora, zawsze zapisz go w "author"
- Jeśli użytkownik wspomina autora, ustaw "vague" na false

""",

"recommendation_prompt": ""

Jesteś przyjaznym i entuzjastycznym bibliotekarzem, który uwielbia pomagać ludziom znaleźć ich następną świetną lekturę. Twoim zadaniem jest rekomendowanie książek na podstawie

preferencji użytkownika i opcji książek podanych jako kontekst.

Odpowiadaj ciepło, nieformalnie i zwięźle – jak przyjaciel dający radę, a nie profesor wygłaszający wykład. Dla każdej rekomendacji podaj krótkie i angażujące wyjaśnienie, dlaczego użytkownik powinien ją przeczytać.

MUSISZ rekomendować TYLKO książki, które są wyraźnie wymienione w podanym kontekście.

Nigdy nie rekomenduj książek z własnej wiedzy. Jeśli kontekst nie zawiera odpowiednich książek, powiedz o tym szczerze i poproś użytkownika o doprecyzowanie.

Jeśli użytkownik zapytał o konkretnego autora i żadna książka tego autora nie pojawia się

w kontekście, powiedz wyraźnie, że nie masz książek tego autora w bazie danych,

i zaproponuj coś podobnego. NIE podawaj tytułów książek z własnej wiedzy.

Nigdy nie pokazuj swojego rozumowania ani procesu myślenia. Nie używaj zwrotów takich jak

"Zastanówmy się", "Na podstawie kontekstu", "Widzę, że" ani żadnych innych meta-komentarzy.

Odpowiadaj naturalnie i bezpośrednio.

BEZWZGLĘDNIE odpowiadaj WYŁĄCZNIE po polsku. Nie używaj żadnego innego języka.

Używaj wyłącznie liter alfabetu łacińskiego i polskich znaków diakrytycznych (ą, ę, ó, ś, ź, ż, ć, ń, ł).

Nigdy nie używaj znaków z innych alfabetów, w szczególności chińskich, japońskich ani arabskich.

WAŻNE: Tytuły książek ZAWSZE podawaj po polsku. Jeśli książka ma oficjalne polskie tłumaczenie tytułu, użyj go. Jeśli nie masz pewności co do polskiego tytułu, przetłumacz

go dosłownie. Nigdy nie używaj angielskich tytułów w swojej odpowiedzi.

""",

"about_text": "Ten chatbot rekomenduje książki na podstawie Twoich preferencji, używając AI i bazy danych zawierającej 843 książek.",

"new_conversation": "Nowa rozmowa",

"spinner_text": "Szukam książek dla Ciebie...",

"input_placeholder": "Co chciałbyś przeczytać?",

"not_found_message": "Przepraszam, wygląda na to, że **{book}** nie ma w mojej bazie danych. Czy chciałbyś, żebym zaproponował coś na podstawie Twoich innych preferencji, lub mógłbyś opisać, co podobało Ci się w tej książce?",

"error_search": "Coś poszło nie tak z wyszukiwaniem. Spróbuj ponownie.",

"error_response": "Coś poszło nie tak. Spróbuj ponownie.",

}

}

Źródło: kod własny

LANGUAGES zawiera instrukcje do ekstrakcji informacji z wektorowej bazy danych i do rekomendacji książek. W przypadku ekstrakcji informacji, model ma się wcielić w asystenta w systemie polecającym książki użytkownikom. Asystent ma zwrócić obiekt typu json, zawierający nastrój utworu, gatunki, wątki literackie, tempo narracji, epokę literacką, odwołanie do książki, jeżeli jest, imię i nazwisko autora oraz wskaźnik vague. Wspomniany wskaźnik ma domyślnie wartość false, jednak jeżeli zapytanie użytkownika okaże się zbyt powierzchowne, aby wyciągnąć z niego istotne informacje, i nie będzie zawierało imienia autora, wartość wskaźnika zostanie ustawiona na false. Asystent ma zwracać tylko obiekt typu json, ustawiać wartość vague na false tylko gdy nie potrafi wyciągnąć informacji pomocniczych z wiadomości użytkownika oraz zachowywać wspomniany przez użytkownika utwór w zmiennej book_reference. W przypadku rekomendacji, rola dla modelu językowego to przyjazny i entuzjastyczny bibliotekarz, który lubi pomagać ludziom szukać nowej książki do przeczytania. Rekomendacje mają być zwięzłe, o nieformalnym, przyjaznym charakterze. Bibliotekarz ma także krótko wyjaśniać wybór polecanego utworu. Model może polecać wyłącznie utwory znajdujące się w bazie danych. W przypadku gdy utworu, do którego odwołał się użytkownik w swojej wiadomości, nie istnieje w wektorowej bazie danych, bibliotekarz ma przeprosić i zaproponować polecenie na podstawie innych preferencji użytkownika. W zależności od tego, jaki język został wybrany w interfejsie chatbota, odpowiedzi mają być zwracane wyłącznie w języku polskim lub w języku angielskim. W przypadku z językiem polskim, model ma korzystać z polskich znaków diakrytycznych i zawsze zwracać nazwę utworu w języku polskim. Jeżeli książka ma oficjalną polską nazwę, model ma ją użyć. W przeciwnym przypadku nazwa zostanie dosłownie przetłumaczona z języka angielskiego.

W kolejnej części kodu zostaje zdefiniowany wygląd chatbota:

Rysunek 13. Interfejs chatbota

```
st.set_page_config(page_title="LitMatch", page_icon="📖")

st.markdown("""

    <style>

        .stApp {

            background-color: #99ccff;
```

```
}

[data-testid="stSidebar"] {

  background-color: #ffffff;

  border-right: 1px solid #e0e0e0;

  font-family: Helvetica, Arial, sans-serif;

}

[data-testid="stSidebar"] p {

  font-family: Helvetica, Arial, sans-serif;

  color: #444444;

  font-size: 0.9rem;

}

h1 {

  font-family: Helvetica, Arial, sans-serif;

  color: #1a1a2e;

  font-size: 2.2rem;

  font-weight: 700;

  letter-spacing: -0.5px;

}

[data-testid="stChatMessage"]:has([data-testid="stChatMessageAvatarUser"])
{

  background-color: #d6e4f7;

  border-radius: 12px;

  padding: 8px;

}
```

```
margin: 4px 0;

    }

    [data-testid="stChatMessage"]:has([data-
testid="stChatMessageAvatarAssistant"]) {

        background-color: #ffffff;

        border-radius: 12px;

        padding: 8px;

        margin: 4px 0;

        border: 1px solid #d0d9e8;

    }

    [data-testid="stChatInput"]:focus-within {

        border-color: #a8bfe0 !important;

        box-shadow: 0 0 0 2px rgba(99, 149, 210, 0.3) !important;

    }

    [data-testid="stChatInput"] {

        border-radius: 12px;

        border: 1px solid #d0d9e8;

        background-color: #ffffff;

        box-shadow: none;

    }

    [data-testid="stBottom"] {

        background-color: #99ccff;
```

```
    box-shadow: none;

border: none;

}

.stBottom > div {

    background-color: #99ccff;

    box-shadow: none;

    border-top: none;

}

.footer {

    position: fixed;

    bottom: 8px;

    left: 50%;

    transform: translateX(-50%);

    font-family: Helvetica, Arial, sans-serif;

    font-size: 0.75rem;

    color: #1a1a2e;

    opacity: 0.6;

    white-space: nowrap;

    z-index: 999;

}

.footer {

    position: fixed;
```

```

bottom: 8px;

    left: 50%;

        transform: translateX(-50%);

        font-family: Helvetica, Arial, sans-serif;

        font-size: 0.9rem;

        color: #1a1a2e;

        opacity: 0.6;

        white-space: nowrap;

        z-index: 999;

    }

    .uwb-logo {

        position: fixed;

        top: 70px;

        right: 16px;

        width: 110px;

        opacity: 0.85;

        mix-blend-mode: multiply;

        z-index: 999;

    }

</style>

"", unsafe_allow_html=True)

st.title("📖 LitMatch")

```

```
def get_image_base64(path):
    with open(path, "rb") as f:
        return base64.b64encode(f.read()).decode()

logo_b64 = get_image_base64("C:/Users/m.novikov/diploma_project/uwb_logo.jpg")

st.markdown(
    f'',
    unsafe_allow_html=True
)

st.markdown(
    '<div class="footer">© Mykyta Novikov, Informatyka, 3. rok, Filia UwB w Wilnie, 2026</div>',
    unsafe_allow_html=True
)

with st.sidebar:
    if "selected_language" not in st.session_state or
    st.session_state.get("selected_language") == "English":
        st.markdown("**Languages**")
    else:
        st.markdown("**Języki**")

    selected_language = st.selectbox(
        "",
        options=list(LANGUAGES.keys()),
```

```

        key="selected_language"

    )

    lang = LANGUAGES[selected_language]

    st.markdown("### About" if selected_language == "English" else "### O
aplikacji")

    st.write(lang["about_text"])

    if st.button(lang["new_conversation"]):

        st.session_state.conversation_history = [{"role": "system", "content":
lang["recommendation_prompt"]}]]

        st.session_state.display_history = []

        st.rerun()

if "conversation_history" not in st.session_state:

    st.session_state.conversation_history = [{"role": "system", "content":
lang["recommendation_prompt"]}]]

if "display_history" not in st.session_state:

    st.session_state.display_history = []

for message in st.session_state.display_history:

    with st.chat_message(message["role"]):

        st.write(message["content"])

```

Źródło: kod własny

Zgodnie z powyższym kodem, okno chatbota w przeglądarce będzie miało ikonkę otwartej książki i nazwę LitMatch. Główna część strony będzie miała niebieski kolor oraz nagłówek LitMatch z ikonką książki. W dolnej części strony, także o niebieskim kolorze, będzie się znajdowało pole do wprowadzenia zapytań użytkownika. Powiadomienia chatbota będą miały biały kolor i zaokrąglone kąty, wtedy jak powiadomienia użytkownika będą miały jasnoniebieski kolor. Kontekst konwersacji jest przechowywany w kluczu `conversation_history`, a historia wyświetlacza – w kluczu `display_history` w `st.session_state`. Jeżeli któregoś z kluczy nie istnieje, zostanie utworzony. Strona będzie także miała panel boczny z tekstem wyjaśniającym przeznaczenie chatbota. W panelu bocznym znajdzie się także przycisk czyszczący historię konwersacji oraz lista pozwalająca na wybór języka. Kliknięcie opcji spowoduje przełączenie się chatbota na wybrany język. Najważniejszą częścią pliku `query_chatbot.py` jest mechanizm pytania-odpowiedzi, który przedstawia poniższy rysunek:

Rysunek 14. Mechanizm pytania-odpowiedzi w pliku `query_chatbot.py`

```
chroma_client =  
chromadb.PersistentClient(path="C:/Users/m.novikov/diploma_project/chroma_db")  
  
collection = chroma_client.get_collection(name="books")  
  
user_input = st.chat_input(lang["input_placeholder"])  
  
if user_input:  
    with st.chat_message("user"):  
        st.write(user_input)  
        st.session_state.display_history.append({"role": "user", "content": user_input})  
  
    try:  
        extraction = client.chat.completions.create(  
            messages=[
```

```
{"role": "system", "content": lang["extraction_prompt"]},

{"role": "user", "content": user_input}

],

model="meta-llama/llama-4-scout-17b-16e-instruct",

)

extraction_response = extraction.choices[0].message.content

extraction_response =
extraction_response.strip().removeprefix("```json").removeprefix("```").removesuffix("```").strip()

params = json.loads(extraction_response)

except Exception as e:

params = {"vague": True, "reference_book": None, "author": None}

if params.get("reference_book"):

original_title = params["reference_book"]

if selected_language == "Polski":

try:

translation = client.chat.completions.create(

messages=[

{"role": "system", "content": """"You are a book title translator. If the given
text is a book title in Polish, translate it to English and

return ONLY the English title. If the title is already in English or is a proper
name that doesn't need translation, return it exactly

as given. Never explain, never add anything else.""""},

{"role": "user", "content": params["reference_book"]}]
```

```
],

model="meta-llama/llama-4-scout-17b-16e-instruct",

)

params["reference_book"] = translation.choices[0].message.content.strip()

except:

pass

reference_check = collection.query(

query_texts=[params["reference_book"]],

n_results=1

)

top_result = reference_check["metadatas"][0][0]["title"].lower()

reference_title = params["reference_book"].lower()

distance = reference_check["distances"][0][0]

if (reference_title not in top_result and top_result not in reference_title) or
distance > 1.0:

not_found_message = lang["not_found_message"].format(book=original_title)

with st.chat_message("assistant"):

st.write(not_found_message)

st.session_state.display_history.append({"role": "assistant", "content":
not_found_message})

st.session_state.conversation_history.append({"role": "assistant", "content":
not_found_message})

st.stop()

if params.get("vague"):

vague_response = "Chętnie pomogę Ci znaleźć coś do czytania! Czy mógłbyś
powiedzieć mi trochę więcej o swoich preferencjach? Na przykład, jaki gatunek
```

```
lubisz, jaki nastrój Cię interesuje, albo czy jest jakaś książka, którą już
czytałeś i podobała Ci się?" if selected_language == "Polski" else "I'd love to

help you find something to read! Could you tell me a bit more about your
preferences? For example, what genre do you enjoy, what mood are you in the mood
for, or is there a book you've already read and enjoyed?"

with st.chat_message("assistant"):

st.write(vague_response)

st.session_state.display_history.append({"role": "assistant", "content":
vague_response})

st.session_state.conversation_history.append({"role": "assistant", "content":
vague_response})

st.stop()

query_parts = []

if params.get("author"):

query_parts.append(f"Author: {params['author']}")

if params.get("mood"):

query_parts.append(f"Mood: {params['mood']}")

if params.get("genres"):

query_parts.append(f"Genres: {' , '.join(params['genres'])}")

if params.get("themes"):

query_parts.append(f"Themes: {' , '.join(params['themes'])}")

if params.get("pacing"):

query_parts.append(f"Pacing: {params['pacing']}")

if params.get("literary_period"):

query_parts.append(f"Literary period: {params['literary_period']}")
```

```

if params.get("reference_book"):

    query_parts.append(f"Similar to: {params['reference_book']}")

    enriched_query = " ".join(query_parts) if query_parts else user_input

    try:

        query_results = collection.query(

            query_texts=[enriched_query],

            n_results=5

        )

    except Exception as e:

        st.error(lang["error_search"])

        st.stop()

    query_results_formatted = []

    for book in query_results["metadatas"][0]:

        query_results_formatted.append(book)

    context = "\n".join([str(book) for book in query_results_formatted])

    author_note = f"\nNote: the user is specifically looking for books by {params.get('author')}." if params.get("author") else ""

    user_message = f"Here are some relevant books:\n{context}\n\nUser request: {user_input}{author_note}"

    st.session_state.conversation_history.append({"role": "user", "content": user_message})

```

```
with st.spinner(lang["spinner_text"]):

try:

chat_completion = client.chat.completions.create(

messages=st.session_state.conversation_history,

model="meta-llama/llama-4-scout-17b-16e-instruct",

)

response = chat_completion.choices[0].message.content

if selected_language == "Polski":

try:

title_fix = client.chat.completions.create(

messages=[

{"role": "system", "content": ""

You are a Polish language editor.

Your task is to find any book titles in English in the given text and replace them

with their Polish translations.

If a book has an official Polish title, use it. If not, translate literally.

Return the corrected text only, with no explanation or additional content.

Keep everything else exactly the same – only translate book titles.

Use only Latin alphabet and Polish diacritical marks (ą, ę, ó, ś, ź, ż, ć, ń, ł).

Never use characters from other alphabets.

""},

{"role": "user", "content": response}

],

model="meta-llama/llama-4-scout-17b-16e-instruct",

)

response = title_fix.choices[0].message.content.strip()
```

```
except:

pass

except Exception as e:

st.error(lang["error_response"])

st.stop()

with st.chat_message("assistant"):

st.write(response)

st.session_state.conversation_history.append({"role": "assistant", "content":
response})

st.session_state.display_history.append({"role": "assistant", "content":
response})
```

Źródło: kod własny

Zanim mechanizm zostaje uruchomiony, z wektorowej bazy danych pobiera się kolekcję books, która przechowuje informacje o książkach, a z pola wejścia pobiera się zapytanie użytkownika. Dalsza część kodu zostaje uruchomiona wyłącznie w przypadku, gdy użytkownik rzeczywiście wpisał pytanie w pole do tego przeznaczone. Na pierwszym etapie z zapytania użytkownika zostaną wyciągnięte kluczowe parametry, które pozwolą modelowi językowemu na dobór rekomendacji. Zapytanie zostanie zapisane po stronie użytkownika w słowniku display_history. Zarówno do ekstrakcji parametrów, jak i do rekomendacji zostanie użyty wielki model językowy Llama 4 Scout. W przypadku ekstrakcji otrzyma on zapytanie zawarte w kluczu extraction_prompt ze słownika LANGUAGES oraz wiadomość od użytkownika. Odpowiedź modelu zostaje zapisana do zmiennej extraction_response i poddana transformacji, celem której jest usunięcie niepożądanych znaków. Później, z extraction_response do zmiennej params w formacie JSON zostaną załadowane parametry książki. Jeżeli modelowi nie udało się wyciągnąć parametrów z zapytania użytkownika, w

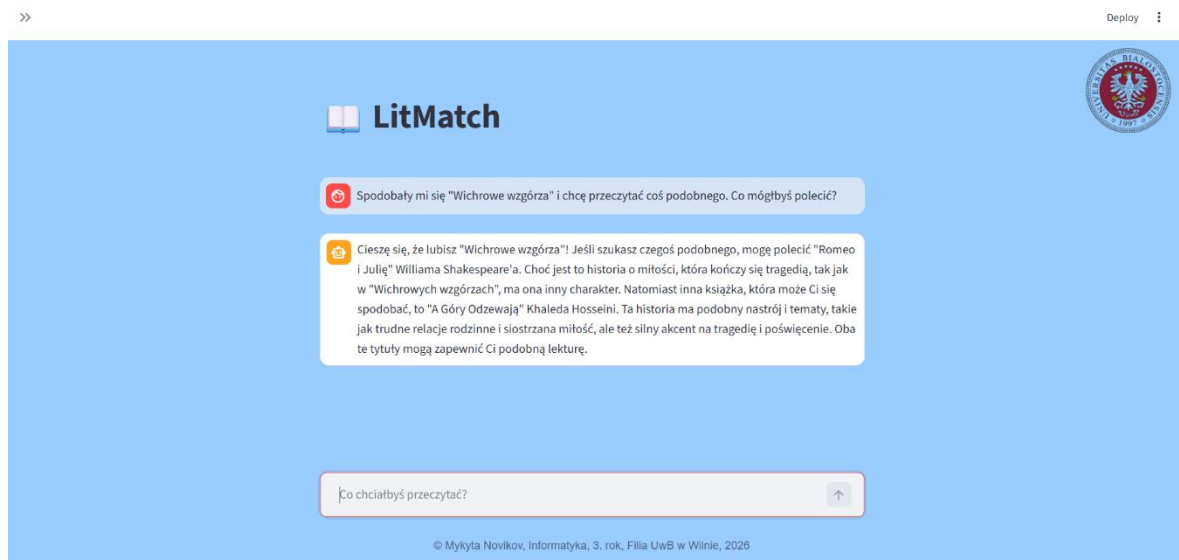
słowniku params powstaną trzy klucze: `vague` z wartością `True` oraz `author` i `reference_book` z wartością `None`. Jeżeli użytkownik wybrał język polski do komunikacji z chatbotem i odwołał się do książki w swojej wiadomości, model językowy przetłumaczy polską nazwę na język angielski. Następnie, jeżeli w odpowiedzi modelu pojawia się wskaźnik `vague`, zostanie wyświetlona wiadomość prosząca użytkownika o podanie szczegółów. W przeciwnym przypadku, do listy `query_parts` zapisuje się parametry, które udało się wyciągnąć z wiadomości użytkownika. Listę później dodaje się do zmiennej `enriched_query`. Zawartość `enriched_query` wejdzie później do zapytania, które zostanie wykonane na kolekcji `books` i ma zwrócić pięć utworów. Na ostatnim etapie utwory z wyniku zapytania na kolekcji `books` zostają dodane do listy `query_results_formatted`. Następnie, na podstawie zawartości `query_results_formatted` powstanie zmienna `context` z tytułami książek. Treść `context` zostanie wstawiona do ciągu `user_message` razem z wartością zmiennej `user_input`. Do `user_message` zostanie także dodana zawartość zmiennej `author_note`, która pokazuje, czy użytkownik w swojej wiadomości odwoływał się do konkretnego autora. Wartość `user_message` zostanie zapisana w `conversation_history`, pamięci modelu dla danej sesji. Po zapisie wielki model językowy ma sformułować rekomendacje na podstawie treści `conversation_history`. Odpowiedź chatbota zostanie zapisana do zmiennej `response`, a potem wyświetlona w rozmowie ze strony chatbota. Jeżeli użytkownik wybrał język polski do rozmowy z chatbotem, model językowy będzie musiał sprawdzić, czy w rekomendacji występują angielskie nazwy utworów. Jeżeli tak, ma je przetłumaczyć na język polski. Ostatnie krokiem jest zapis rekomendacji do `conversation_history` i `display_history` dla roli asystenta.

ROZDZIAŁ 4. TESTOWANIE

4.1 Rekomendacja na podstawie tytułu

Pierwszym testem, któremu poddano chatbot, sprawdza reakcję na zapytanie z odwołaniem wyłącznie do tytułu utworu. W ramach testu ze strony użytkownika wysłano wiadomość z prośbą o polecenie utworu podobnego do „Wichrowych wzgórz” autorstwa Emily Brontë. Chatbot udzielił odpowiedzi.

Rysunek 15. Test - rekomendacja na podstawie tytułu

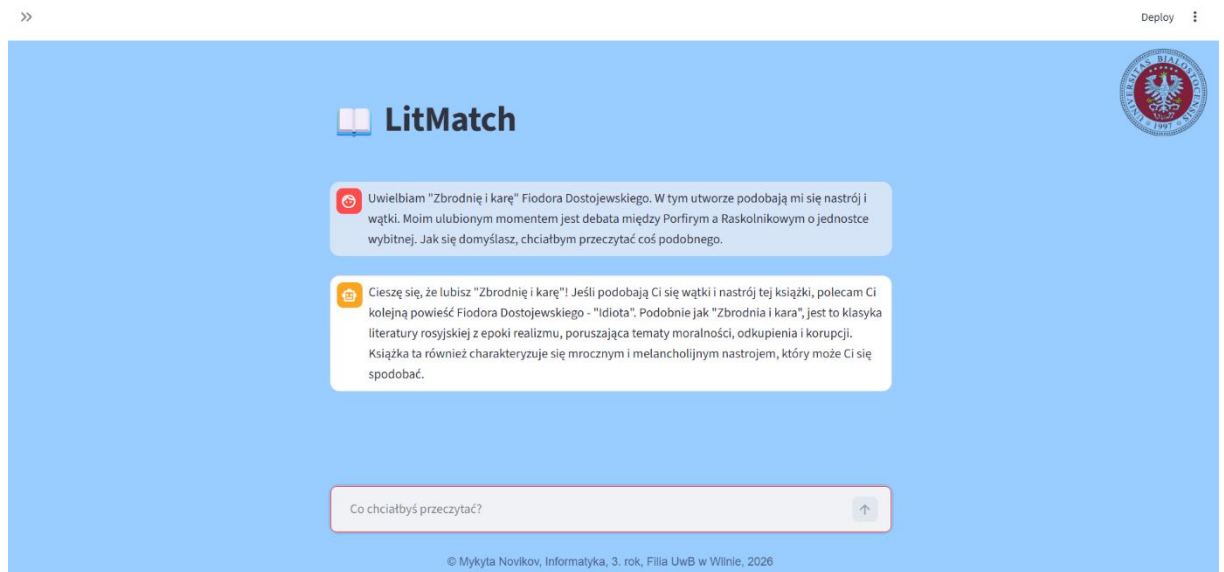


Źródło: kod własny uruchomiony w przeglądarce Microsoft Explorer

4.2 Rekomendacja na podstawie tytułu i opisu

Następny test, który został wykonany na chatbocie, sprawdza jego odpowiedź w sytuacji, gdy użytkownik w swojej wiadomości odwołuje się do książki i podaje elementy, które mu się podobały. W ramach testu ze strony użytkownika wysłano wiadomość z odwołaniem do utworu „Zbrodnia i kara” Fiodora Dostojewskiego oraz opisem tego, co się spodobało użytkownikowi w utworze. Chatbot udzielił odpowiedzi, podając rekomendacje i wyjaśnienia.

Rysunek 16. Test chatbota - rekomendacja na podstawie tytułu i opisu

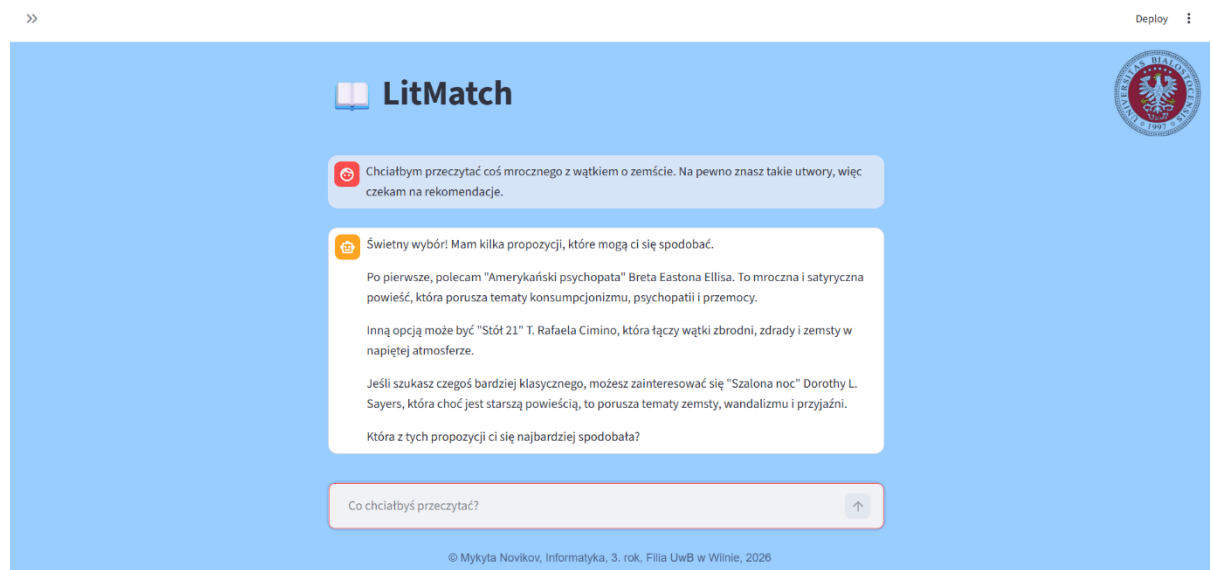


Źródło: kod własny uruchomiony w przeglądarce Microsoft Explorer

4.3 Rekomendacja na podstawie opisu

Trzeci test chatbota sprawdza czy potrafi polecić utwory, jeżeli użytkownik podaje tylko opis elementów, które mu się podobają w książkach, nie podając przy tym tytułów. W ramach testu ze strony użytkownika wysłano prośbę o polecenie czegoś „mrocznego z wątkiem o zemście”.

Rysunek 17. Test - rekomendacja na podstawie opisu

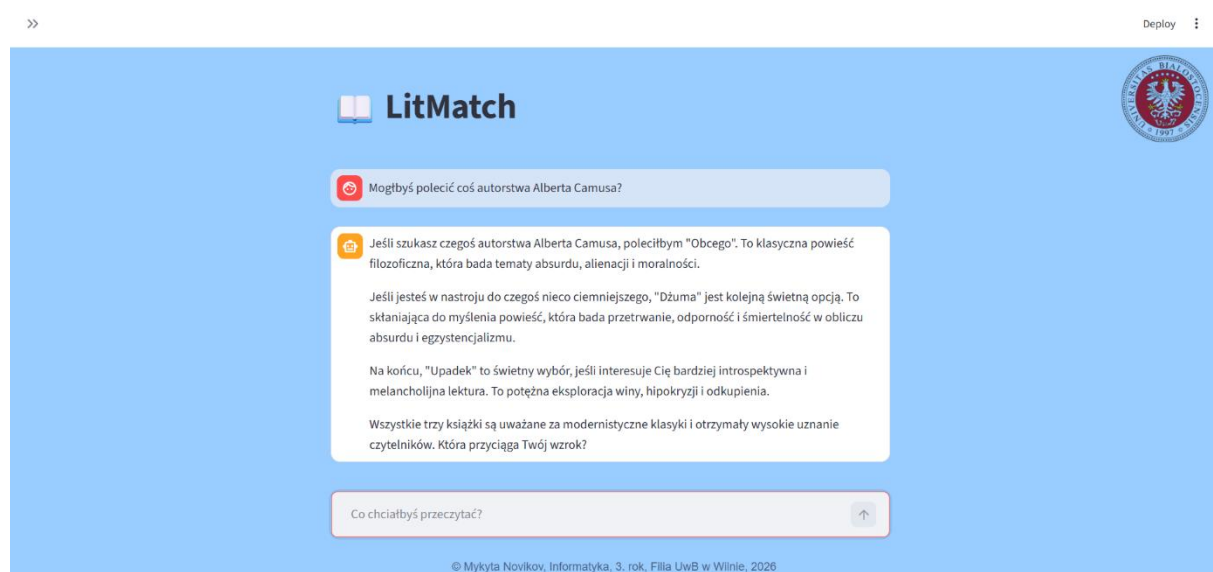


Źródło: kod własny uruchomiony w przeglądarce Microsoft Explorer

4.4 Rekomendacja na podstawie autora

Następny test sprawdzono zdolność chatbota do polecenia utworów na podstawie odwołania do autora. W ramach testu została wysłana wiadomość z prośbą o rekomendacje utworów Alberta Camusa. Chatbot odpowiedział, oferując użytkownikowi trzy pozycje do przeczytania.

Rysunek 18. Test - rekomendacja na podstawie autora

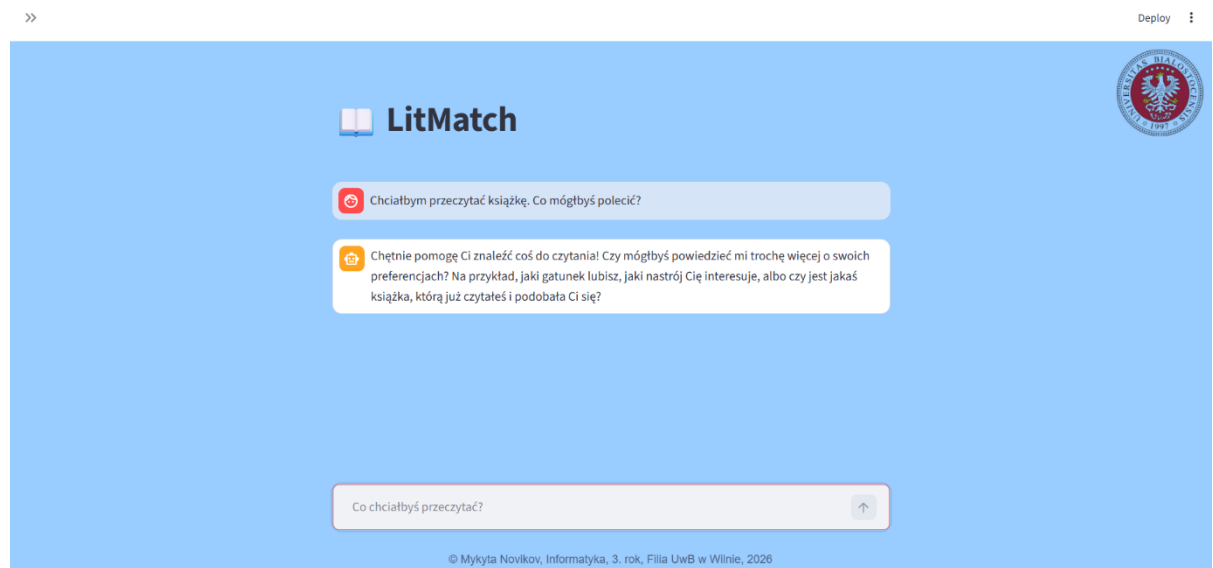


Źródło: kod własny uruchomiony w przeglądarce Microsoft Explorer

4.5 Powierzchowne zapytanie

W ramach kolejnego testu do chatbota wysłano zapytanie: „Chciałbym przeczytać książkę. Co mógłbyś polecić?”. Zapytanie nie zawiera informacji pozwalających na utworzenie rekomendacji, więc w swojej odpowiedzi chatbot poprosił o podanie szczegółów, na podstawie których można byłoby wnioskować o preferencjach literackich osoby.

Rysunek 19. Test chatbota - powierzchowne zapytanie

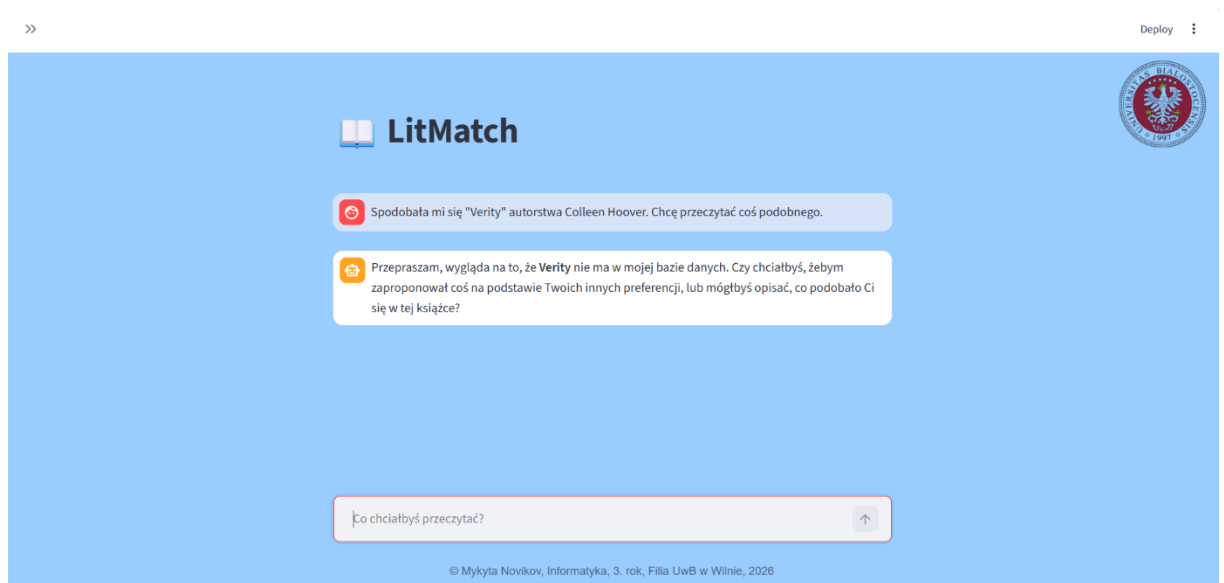


Źródło: kod własny uruchomiony w przeglądarce Microsoft Explorer

4.6 Utwór spoza bazy danych

Następny test sprawdzał przestrzeganie ważnej zasady: nie dawać rekomendacji, jeżeli użytkownik podał nazwę utworu, którego nie ma w bazie danych wykorzystywanej przez chatbot. W ramach testu wysłano wiadomość z odwołaniem do książki „Verity” autorstwa Colleen Hoover. Ponieważ danej pozycji nie ma wśród przechowywanych w bazie danych, chatbot w udzielonej odpowiedzi przeprosił i wyjaśnił sytuację, oferując pomoc, jeżeli użytkownik poda inny utwór lub szczegóły dotyczące tego, co mu się spodobało w podanym utworze.

Rysunek 20. Test chatbota - utwór spoza bazy danych

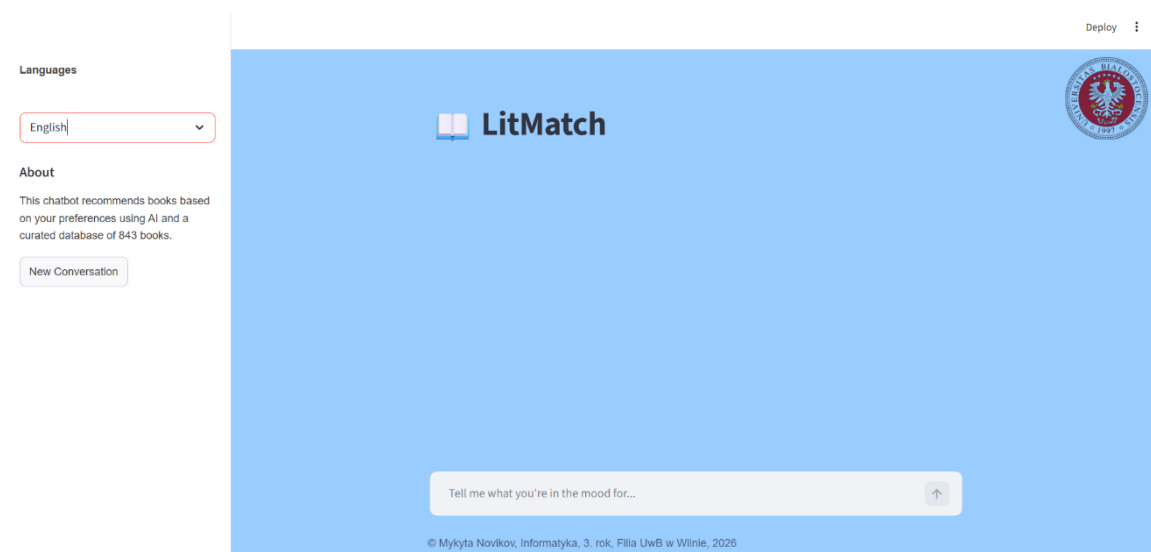


Źródło: kod własny uruchomiony w przeglądarce Microsoft Explorer

4.7 Weryfikacja języka angielskiego

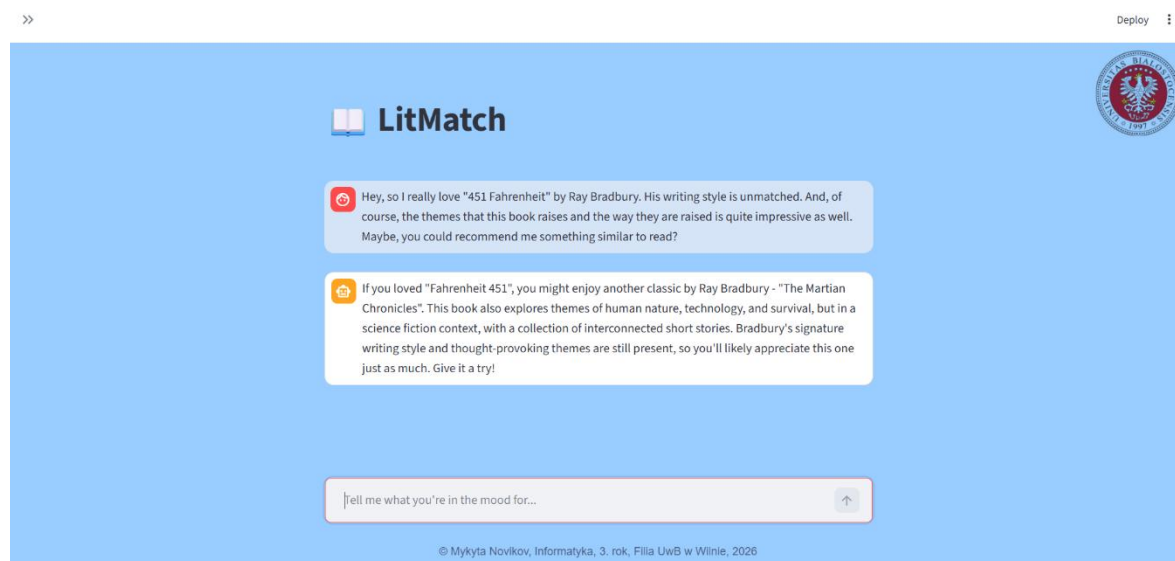
W ostatnim teście przetestowano działanie chatbota w języku angielskim. W tym celu z listy w panelu bocznym wybrano język angielski. Napisy, które początkowo były w języku polskim, zostały przetłumaczone. Do chatbota później wysłano wiadomość w języku angielskim, proszącą o rekomendacje z odwołaniem do utworu „451 stopni Fahrenheita”. Odpowiedź została udzielona również w języku angielskim.

Rysunek 21. Test - interfejs chatbota w języku angielskim



Źródło: kod własny uruchomiony w przeglądarce Microsoft Explorer

Rysunek 22. Test - działanie chatbota w języku angielskim



Źródło: kod własny uruchomiony w przeglądarce Microsoft Explorer

ZAKOŃCZENIE

Prototyp chatbota polecającego książki na podstawie preferencji użytkownika został zrealizowany. Chatbot używa kolekcji przechowującej informacje o książkach w wektorowej bazie danych Chroma oraz modelu językowego Llama 4 Scout. Wielki model językowy jest wykorzystywany do wzbogacenia informacji o książkach, ekstrakcji parametrów z wiadomości użytkownika, tłumaczenia polskich nazw utworów na język angielski i generowania rekomendacji. Stworzony prototyp pozwala na komunikację w języku polskim i angielskim. Przełączenie się między językami odbywa się za pomocą listy w panelu bocznym, który zawiera także przycisk „Nowa rozmowa” usuwający wszystkie wiadomości z rozmowy. Aby otrzymać rekomendację od chatbota, użytkownik ma podać tytuł utworu, który mu się podoba, albo opisać swoje preferencje dotyczące atrybutów dzieła, np. tempa narracji czy wątków literackich. W przypadku odwołania się przez użytkownika do utworu, którego nie ma z bazy danych wykorzystywanej przez prototyp, zostanie wyświetlona wiadomość wyjaśniająca sytuację. Chatbot proponuje wybrać inną książkę lub opisać, co się spodobało użytkownikowi w książce, o której wspomniał wcześniej. Wszystkie wspomniane wyżej funkcjonalności działają sprawnie.

Planowana jest kontynuacja pracy nad chatbotem. W swojej obecnej wersji korzysta on z przechowywanej lokalnie wektorowej bazy danych oraz skryptu, który zawiera kluczową logikę i interfejs użytkownika. To sprawia, że LitMatch jest ograniczony do urządzenia, na którym został uruchomiony, i inni użytkownicy nie mają do niego dostępu. W celu uniezależnienia aplikacji od urządzenia lokalnego można skorzystać z chmury. Wektorową bazę danych można byłoby przenieść do zarządzanego serwisu, np. Pinecone, wtedy jak kluczowy kod mógłby zostać przeniesiony i uruchomiony na Streamlit Cloud lub podobnej platformie do hostowania aplikacji Streamlit. Wspomniane wyżej zmiany pozwolą zamienić LitMatch w aplikację globalną, która nie jest ograniczona do pojedynczego urządzenia.

Kolejnym obszarem, w którym jest możliwe polepszenie, to automatyzacja przetwarzania danych. Obecnie, w przypadku dodania nowego rekordu do bazy danych PostgreSQL należy ręcznie uruchomić skrypty do wzbogacenia danych, przemiany danych na wektory i aktualizacji aplikacji. Proces ten można byłoby usprawnić za pomocą zautomatyzowanego potoku, który byłby uruchamiany przez zmiany w bazie danych PostgreSQL. Taki potok mógłby automatycznie uruchamiać mechanizmy wzbogacenia danych, ich przemiany na wektory i aktualizowałby aplikację bez potrzeby ingerencji człowieka.

Kolejna zmiana, która przyniosłaby korzyść, to dodanie możliwości tworzenia kont dla użytkowników. Pozwoliłoby to na polepszenie działania mechanizmu rekomendacji, ponieważ użytkownicy mogliby zostawiać opinie o rekomendacjach otrzymanych od chatbota. Na podstawie opinii LitMatch mógłby lepiej się uczyć preferencji i dopasowywać rekomendacje w taki sposób, aby lepiej odpowiadały gustom osoby zadającej pytanie.

Ostatnią zmianą, której można byłoby dokonać, to dodanie klikalnych linków do rekomendacji chatbota. Kliknięcie linku będzie przekierowywało użytkownika na strony

internetowe, gdzie będzie mógł kupić polecane utwory bądź przeczytać o nich więcej. Ta zmiana zrobiłaby chatbot bardziej praktycznym i użytecznym.

SPIS RYSUNKÓW

Rysunek 1. Plik create_table.py	14
Rysunek 2. Plik load_data.py	16
Rysunek 3. Import modułów i funkcji w pliku enrich_data.py	18
Rysunek 4. Zapytanie w pliku enrich_data.py	19
Rysunek 5. Połączenie z bazą danych i pobranie informacji o książkach w pliku enrich_data.py	21
Rysunek 6. Mechanizm wzbogacenia informacji o książkach z pliku enrich_data.py	22
Rysunek 7. Import modułów i funkcji w pliku book_to_vector.py	23
Rysunek 8. Nawiązanie połączenia z bazą danych i pobranie informacji w pliku book_to_vector.py ..	24
Rysunek 9. Normalizacja treści data_dict	25
Rysunek 10. Utworzenie wektorowej bazy danych	27
Rysunek 11. Import modułów i funkcji w pliku query_chatbot.py	27
Rysunek 12. Słownik LANGUAGES w pliku query_chatbot.py	28
Rysunek 13. Interfejs chatbota	34
Rysunek 14. Mechanizm pytania-odpowiedzi w pliku query_chatbot.py	41
Rysunek 15. Test - rekomendacja na podstawie tytułu	49
Rysunek 16. Test chatbota - rekomendacja na podstawie tytułu i opisu	50
Rysunek 17. Test - rekomendacja na podstawie opisu	51
Rysunek 18. Test - rekomendacja na podstawie autora	51
Rysunek 19. Test chatbota - powierzchowne zapytanie	52
Rysunek 20. Test chatbota - utwór spoza bazy danych	53
Rysunek 21. Test - interfejs chatbota w języku angielskim	54
Rysunek 22. Test - działanie chatbota w języku angielskim	54

BIBLIOGRAFIA I ŹRÓDŁA INTERNETOWE

1. AWS, What is RAG (Retrieval-Augmented Generation)? [online], <https://aws.amazon.com/what-is/retrieval-augmented-generation/#how-does-retrieval-augmented-generation-work--1xobboj> [dostęp: 17.05.2026]
2. Cole Stryker, *What are large language models (LLMs)?* [online], <https://www.ibm.com/think/topics/large-language-models> [dostęp: 17.05.2026]
3. Dave Bergman, Cole Stryker, *What is vector embedding?* [online], <https://www.ibm.com/think/topics/vector-embedding> [dostęp: 17.05.2026]
4. Dokumentacja Chroma [online], <https://docs.trychroma.com/docs/overview/getting-started> [dostęp: 11.05.2026].
5. Dokumentacja Groq [online], <https://console.groq.com/docs/quickstart> [dostęp: 11.05.2026].
6. Dokumentacja Python, *ast - Abstract syntax trees* [online], <https://docs.python.org/3/library/ast.html> [dostęp: 11.05.2026].
7. Dokumentacja Python, *json - JSON encoder and decoder* [online], <https://docs.python.org/3/library/json.html> [dostęp: 11.05.2026].
8. Dokumentacja Python, *os - Miscellaneous operating system interfaces* [online], <https://docs.python.org/3/library/os.html> [dostęp: 11.05.2026].
9. Dokumentacja Python, *re - Regular expression operations* [online], <https://docs.python.org/3/library/re.html> [dostęp: 11.05.2026].
10. Dokumentacja pandas, *read_csv* [online], https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html [dostęp: 11.05.2026].
11. Dokumentacja psycopg2 [online], <https://pypi.org/project/psycopg2/> [dostęp: 11.05.2026].
12. Dokumentacja python-dotenv [online], <https://pypi.org/project/python-dotenv/> [dostęp: 11.05.2026].
13. Dokumentacja streamlit [online], <https://docs.streamlit.io/get-started/installation> [dostęp: 11.05.2026].
14. Dokumentacja SQLAlchemy [online], <https://docs.sqlalchemy.org/en/21/core/engines.html> [dostęp: 11.05.2026].
15. Eleni Adamapolou, Lefteris Moussiades, *Chatbots: History, technology, and applications*, ScienceDirect, “Machine Learning with Applications” z 15.12.2020, nr. 2
16. Ishika Johari, *Best Books (10k) Multi-Genre Data* [online], <https://www.kaggle.com/datasets/ishikajohari/best-books-10k-multi-genre-data?resource=download> [dostęp: 11.05.2026].
17. Jeffrey M. Jones, *Americans Reading Fewer Books Than in Past* [online], <https://news.gallup.com/poll/388541/americans-reading-fewer-books-past.aspx> [dostęp: 17.05.2026]

18. Jessica K. Bone, Feifei Bu, Jill K. Sonke, Daisy Fancourt, *The decline in reading for pleasure over 20 years of the American Time Use Survey*, CellPress, “iScience” z 19.09.2025, nr. 9
19. Jonathan Jarry M.Sc., *The ‘Reading Crisis’ in Perspective* [online], <https://www.mcgill.ca/oss/article/critical-thinking/reading-crisis-perspective> [dostęp: 19.05.2026]
20. National Endowment for the Arts (2004), *Reading at Risk: A Survey of Literary Reading in America*, https://www.arts.gov/sites/default/files/RaRExec_0.pdf [dostęp: 19.05.2026]