Book Search with LLM Integration

Agile Technical Design

Sreenath Gopalakrishnan

Contents

1. Introduction

The purpose of this document is to define the specific technology design, technology components, data flow and assumptions made for integrating a book search web application with LLM.
It is intended that this document captures the solution design details pertaining to the deployment of the web application as well as the limitations and assumptions made before starting the project.

2. Problem Statement

As part of SHI's interview process, the interviewer has provided an exercise to design and develop an end-to-end web application and integrate a large language model (LLM) into the application. This includes developing a browser-based user interface, creating backend endpoints, and integrating APIs.
The exercise is to design and develop a browser-based frontend using HTML, CSS and JavaScript to search a book by name and display the matching books retrieved from the backend. The backend should be in Python programming language and expose a book search Web API. The internal book search Web API endpoint should call the external Open Library web service to fetch book data. The data returned from Open Library web service is cumbersome and hard to read by a normal user. Integrating an LLM to process the result with Natural Language Processing capability can simplify the response to more human readable form.

3. Design

We will address the problem statement by building an end-to-end web application where the end user can search for a book by name and get some specific details about the books that matches the search text in friendly readable format. Listed below the features of the web app.
Features:

- User should be able to enter the name of the book in natural language and click a search button to get matching books also in natural language
- System should take the user's input and call a backend Web API service endpoint which return a JSON result with details of matching books
- The internal Web API service should in turn call an external Open Library (https://openlibrary.org/developers) web service with title or the name of the book to get matching JSON data. The Open Library Search API is one of the most convenient and complete ways to retrieve book data on Open Library.
- The Open Library Search API returns a comprehensive list about the book in JSON format, and it is cumbersome for end user to make sense of the data. At the advent of AI and ease of using LLM models for Natural Language Processing, it is better to pass the comprehensive list to an LLM to summarize rather than writing program to scan for relevant information from the JSON data.
- The internal Web API service after receiving the detailed JSON from Open Library Search API should invoke an LLM to summarize the results to easy-to-read natural language list.
- Return the processed results to the frontend and display to end user.

3.1 High-Level Architecture

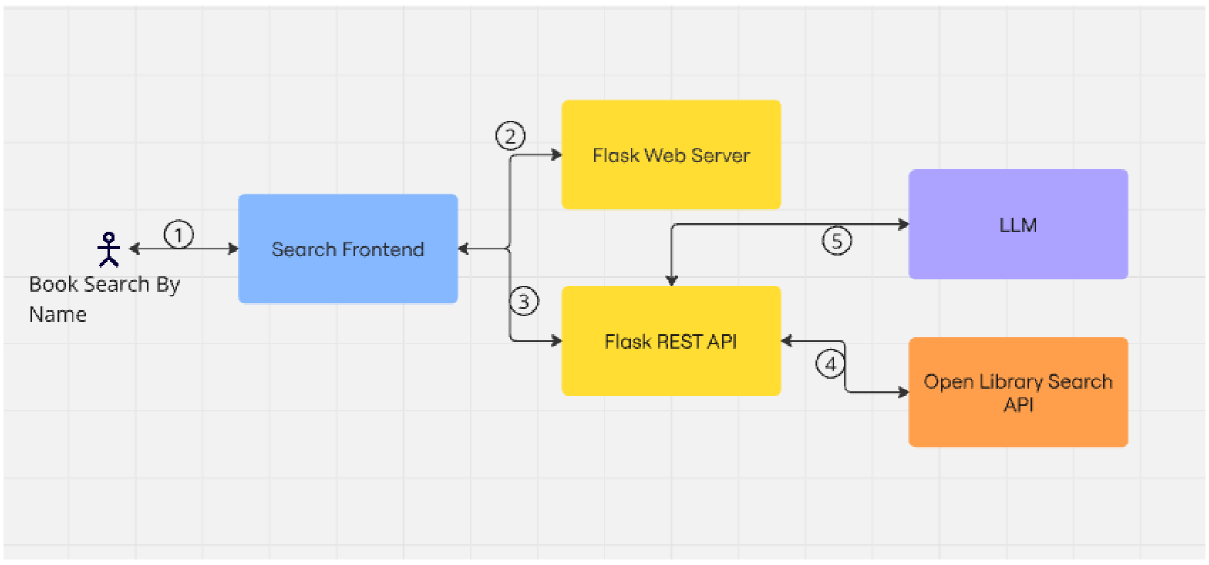
At a high level the application is divided into 5 components

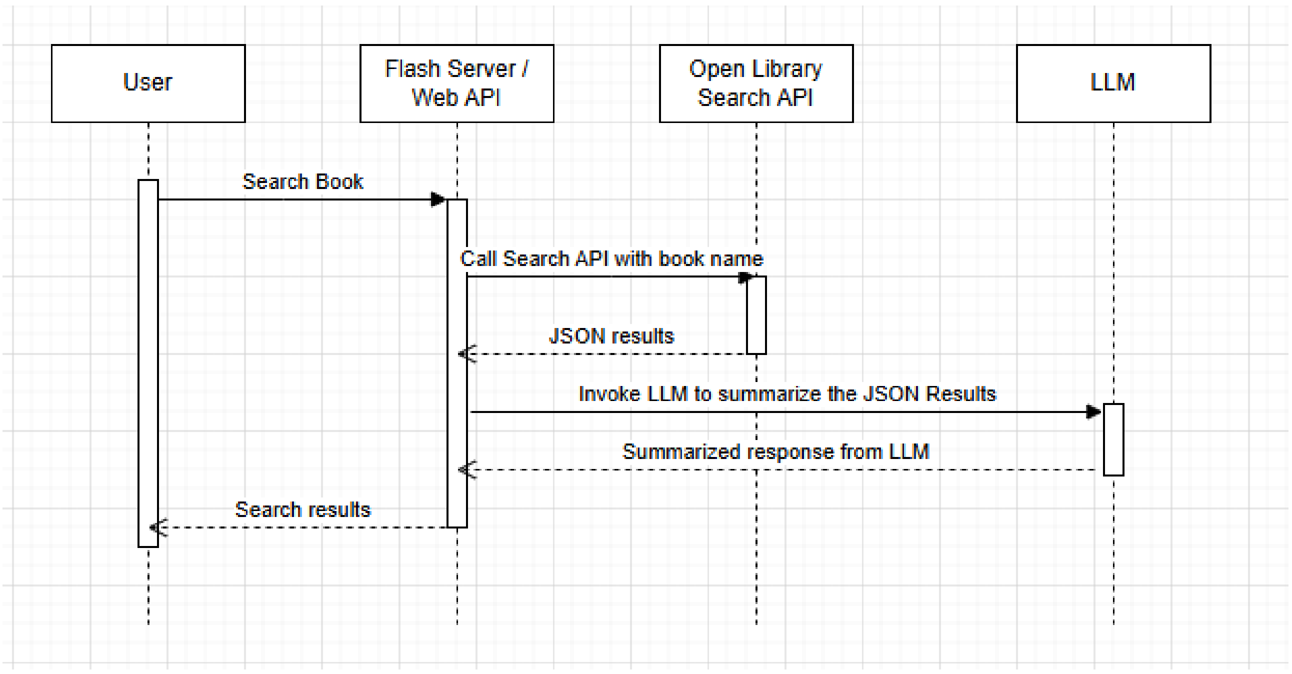
Technical Design

Service	Description	Task	Hosting
Search Frontend	Html web page with a search text box and submit button to get the request search results from the backend	To get the user input and display the search results	Deployed together with he Web app
Flask Web Server	Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications.	Web server to host the web application	Flask is a WSGI application. A WSGI server is used to run the application, converting incoming HTTP requests to the standard WSGI environ, and converting outgoing WSGI responses to HTTP responses. Web Server Gateway Interface (WSGI) is a mediator responsible for conveying communication between a web server and a Python web application. Azure app service Gunicorn: Gunicorn is a pure Python WSGI server with simple configuration and multiple worker implementations for performance tuning.
Flask Web API Endpoint	Python Flask is a popular framework for building web applications and APIs in Python. It provides developers with a quick and easy way to create RESTful APIs that can be used by other software applications. Flask is lightweight and requires minimal setup, making it a great choice for building small to medium-sized APIs	The middleware to receive the search request from the users. Call the Open Library Search API and get the search results. Invoke LLM with the JSON data from the Open Search API	Azure app service Gunicorn: Gunicorn is a pure Python WSGI server with simple configuration and multiple worker implementations for performance tuning.

Service	Description	Task	Hosting
Open Library Search API	External Web Service. The Open Library Search API is one of the most convenient and complete ways to retrieve book data on Open Library. Returns both Work level information about the book (like author info, first publish year, etc.), as well as Edition level information (like title, identifiers, covers, etc.)	Returns search results in JSON format	N/A
LLM	Pre-trained transformer for text generation or summarization. GPT-3.5 (OpenAI) or Llama 2.1 (meta open Source) LangChain: framework that make building llm applications easier. Groq: Platform to allow llama 3.1 in cloud and the inference is very fast.	For Natural Language Processing	N/A

3.1 Architecture Diagram




User
Flash Server / Web API
Open Library Search API
LLM
Search Book
Call Search API with book name
JSON results
Invoke LLM to summarize the JSON Results
Summarized response from LLM
Search results

3.2 Alternatives Considered

As an alternative for Flask Web API the team considered FastAPI. Even though FastAPI is a modern, fast (high-performance), web framework for building APIs with Python it was decided to go with Flask Web API because the Flask is a full framework for building both frontend and backend.

3.3 Outstanding Concerns

As the project have a very tight time timeline, and the end goal of the solution is only for demonstration purpose as well as tools used are free versions with limitations there are some outstanding concerns and those were mitigated with certain requirement adjustments.

i. Limiting the search result count: Sometimes a search to Open Library Search API returns hundreds of results (eg: searching with “lord of the rings” resulted in 3796 results). It is not user friendly to display all these results to the end user. It was decided to limit this to a maximum of 20 and provide provision in the API to receive it from the calling application.

ii. LLM Word limit in a single request: The free version on LLM’s does not support more than 6000 words in a single request. The JSON data returned from the Open Library Search API contains multiple results and each book details contains hundreds of parameters like author_alternative_name, author_key, contributor, cover_edition_key, etc. which might not be relevant for the end user for a simple search to find books that match their search. It was decided to extract only the relevant parameters like title, authors, first_publish_year, etc. for sending to LLM for natural language processing.

iii. Web API Security: Even though the search web page and the web api to process the book search request is served from the same domain it is a best practice to secure the API endpoint using a token-based authentication. It was decided to defer this implementation as this is only for demonstration purposes.

4. Build/Implementation

5. Documentation

6. Mobile Strategy

7. Software, Licencing and Certificates

8. Cost Impact

9. Test Plan

10. Delivery

11. Go-To-Market

12. Appendices/Glossary