Available online at www.fast-forward-it.de/

Company homepage: www.ffit.gmbh

Enhancing User Interaction with Artificial Intelligence

A Liferay Plugin for Large Language Model Integration

Chris Börgermann[1], Kaan Erbay[2]

Fast Forward IT GmbH, Großenbaumer Weg 8, D-40472 Düsseldorf, Germany

ARTICLE INFO

Article history:

Version 1.0
Published 11 November 2024

Keywords:

Liferay DXP
Artificial Intelligence
Large Language Model
Retrieval-Augmented Generation
Semantic Search

ABSTRACT

In this paper, we introduce a Liferay plugin designed to enhance user interaction. Our tool is specifically tailored for product managers and content editors who aim to leverage advanced artificial intelligence capabilities within their Liferay portals. The plugin allows them to feed a large language model (LLM) with content from designated pages and files, creating a comprehensive text corpus. Users can input site-specific questions to an integrated interface and the system then processes these queries by matching them against the pre-defined text tokens, delivering precise and contextually relevant answers.

The architecture is built on a microservices architecture and uses Liferay 7.4 for the user interface and FastAPI for the backend services. Elasticsearch serves as a vector database for semantic search and document management.

Standard Semantic Search vs. Custom Retrieval-Augmented Generation

The integrated semantic search and our retrieval-augmented generation (RAG) plugin each offer unique approaches to information retrieval and processing. The standard semantic search uses natural language processing (NLP) and machine learning to understand the context and meaning of search queries, taking into account synonyms, word variations, and semantic relationships between terms. This method employs algorithms for text analysis and processing to find relevant documents and information. A key advantage in comparison to a simple keyword-matching search is that it delivers more relevant results based on the context and meaning of the query, thereby enhancing the user experience with more accurate and precise search results. It is particularly well-suited for large datasets and diverse sources of information. However, it can face limitations with very specific or complex queries, as the results still rely on existing documents and does not generate answers.

In comparison, our RAG approach combines information retrieval with text-generating AI models. This two-step process first finds relevant documents and then generates a coherent response. The retrieval model searches a database or corpus of documents, while the generation model uses the retrieved information to produce a coherent answer. RAG can provide very precise and specific answers by combining the best information from multiple sources, making it especially suitable for complex queries that require detailed and contextually relevant responses. However, it is more complex to implement, requires more computational and data processing resources, and the generation of responses can be more time-consuming.

In terms of effectiveness, the semantic search is particularly effective for general information retrieval tasks, where a quick and relevant answer to a broad query is needed. RAG, on the other hand, is more effective for complex queries that require detailed and contextually rich answers, especially in dynamic environments or for specific questions.

System Overview

Our system comprises several key components, each playing a vital role in providing a seamless experience for editors managing documents and for users interacting with

[1] cb@fast-forward-it.de

[2] ke@fast-forward-it.de

our customizable chatbot. The main components of our system (see figure 1) include:

(1) Liferay 7.4: Serves as the user interface which allows editors to manage documents and users to interact with the developed chat portlet.
(2) FastAPI Server: Provides three main endpoints for uploading and deleting documents and chatting.
(3) MySQL: Stores uploaded documents.
(4) Elasticsearch: Stores document snippets and their embeddings for semantic search.
(5) Retrieval Augmented Generation (RAG): Combines search results from the vector database and generates answers with an LLM (Large Language Model).

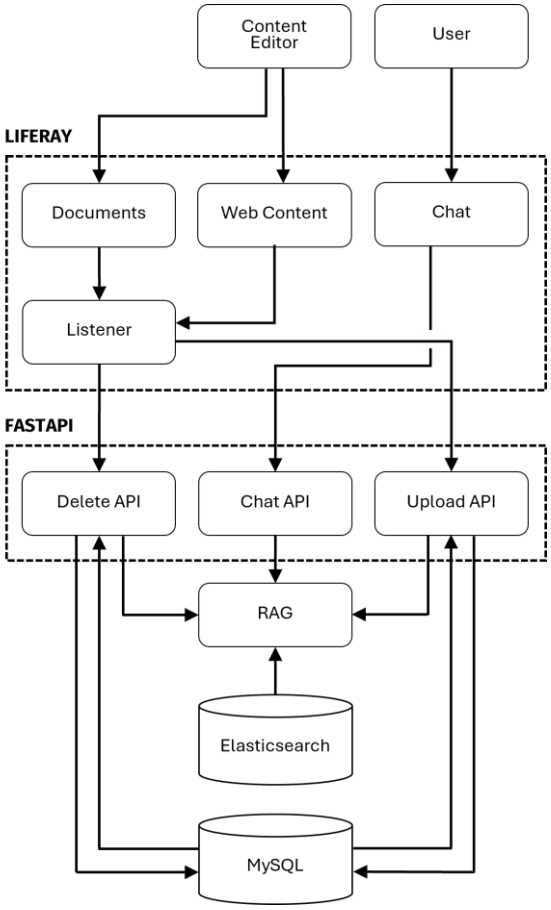

Figure 1: System Architecture

Liferay

Liferay 7.4 serves as a centralized user interface, enabling (1) content editors to upload and delete documents and (2) grant users to interact with a custom chat portlet. This chat portlet integrates with the RAG system, allowing users to receive intelligent responses generated from existing documents.

Document uploads and deletions are tracked by a ModelListener, which interacts with the corresponding REST endpoints of the FastAPI server.

FastAPI

The FastAPI server is the backbone of our system, providing essential API endpoints that manage document uploads, deletions, and user interactions. Each endpoint is designed to perform specific tasks, ensuring a seamless integration between the front-end user interface and the underlying data management and processing systems. The FastAPI server consists of three main endpoints:

(1) Upload API: The Upload API endpoint is responsible for handling the initial ingestion of documents into the system. When a user uploads a document, the API receives a unique FileID along with the file itself. The endpoint calculates an MD5 hash of the uploaded file, which serves as a checksum to ensure data integrity and uniqueness. Along with the file name, this hash is stored in a MySQL database, associating the FileID with these metadata attributes. This information is critical for managing the documents, pro-viding a reference for future operations such as retrieval or deletion. Once the file is received and its metadata is stored, the document is transmitted to the Retrieval Augmented Generation (RAG) system. Here, the file undergoes processing to extract relevant snippets and embeddings.

(2) Delete API: The Delete API endpoint handles the removal of documents from the system. It receives a FileID corresponding to the document that needs to be deleted. The endpoint queries the MySQL database using the FileID to retrieve the associated meta-data, including the file name and other identifiers. Using this information, the system deletes the document entries from both the MySQL database and Elasticsearch. Removing the entries from Elastic-search ensures that the document is no longer retrievable through the semantic search, maintaining the accuracy and relevance of the search results. This step is crucial for managing the lifecycle of documents, ensuring that outdated or irrelevant information is not available to users.

(3) Chat API: The Chat API is the core interface for user interactions. It receives natural language queries from users, which can range from simple factual questions to complex information requests. Upon receiving a query, the Chat API interacts with the RAG system to process the input. This involves using the semantic embeddings stored in Elasticsearch to find relevant document snippets that match the query's intent. The RAG system combines the retrieved snippets with a Large Language Model (LLM), such as GPT-4o mini, to generate a comprehensive and contextually relevant response. [1] [2]

Retrieval-Augmented Generation (RAG)

The Retrieval-Augmented Generation (RAG) process is central to delivering accurate and contextually appropriate responses to user queries in our system. [3] It involves a series of carefully orchestrated steps that transform raw document data into meaningful and retrievable information. The process is divided into two main pipelines: (1) data preparation and (2) data retrieval, followed by the text generation of the overall answer. Each step utilizes advanced technologies and methodologies to ensure quality results.

(1) **Data Preparation Pipeline:** The Data Preparation Pipeline is responsible for processing and structuring the webcontents and uploaded documents to make them suitable for efficient retrieval and search operations. This pipeline includes the following steps: (a) Filehandler: The file handler acts as the initial point of contact for uploaded documents. It identifies the document type (such as text, audio, video, imagery, structured data, markup, source code etc.) and selects the appropriate processing pipeline. Non-text content will be transcribed beforehand. Using Langchain's SemanticChunker, [4] [5] the document is broken down into smaller, semantically coherent chunks. This chunking is crucial for ensuring that each segment of text can be individually understood and indexed, allowing for a more precise retrieval during searches. Semantic chunking helps in capturing the meaning of different parts of the document, which can vary significantly, especially in long or complex documents. (b) Embedding: After chunking, each text segment is embedded using OpenAI's text-embedding-3-large model. This model converts each text into high-dimensional vectors that encapsulate the semantic meaning of the chunks. These embeddings are essential for semantic search, as they allow the system to understand and match the meaning behind words and phrases, rather than just their literal text. (c) Storage: The embeddings, along with the original text and associated metadata (such as document ID, source, and chunk location), are stored in a vector database – Elasticsearch in this case. Storing both the embeddings and the original text enables the system to not only perform semantic searches but also to retrieve and display the exact snippets of text that are most relevant to the user's query.

(2) **Retrieval Pipeline:** The Retrieval Pipeline is activated when a user submits a query. It is designed to find and rank the most relevant document snippets from the database and includes the following steps: (a) Query Rewriting: Using GPT-4o mini first multiple variations of the single user query are generated. Each variation represents a different perspective that might focus on related but distinct aspects or use different terminology. (b) Embedding: The generated queries are embedded using the same text-embedding-3-large model. The embeddings represent the semantic content of the queries, facilitating the matching process with the document embeddings stored in the database. (c) Similarity search: The embedded queries are compared against the document embeddings in Elasticsearch. This search identifies the most semantically similar document chunks, typically retrieving the most relevant documents or snippets. The similarity search leverages the high-dimensional vector space to find documents that best match the meaning of the queries, rather than just the specific words used. The unique union of all the documents is then used to create a larger, more comprehensive set of potentially relevant webcontents and documents (c) Keyword search: Concurrently, GPT-4o mini is employed to extract keywords from the user's query. These keywords are then used to perform a more traditional keyword-based search within the vector database. The keyword search complements the similarity search by ensuring that specific, potentially crucial terms are not overlooked, especially when they are directly relevant to the user's intent. (d) Reciprocal rank function: The results from the similarity search and the keyword search are combined using a method known as Reciprocal Rank Fusion (RRF). This technique merges the ranked lists from both search methods to produce a single, optimized ranking. RRF enhances the final ranking by balancing the strengths of both search approaches. It increases the precision and relevance of the retrieved documents by considering both semantic context and specific keyword matches. (e) Answer generation: The top-ranked document snippets, as determined by the RRF method, are provided as context to a Large Language Model (LLM), specifically GPT-4o mini. GPT-4o mini uses this context to generate a comprehensive and relevant answer to the user query. The model synthesizes information from the provided snippets, producing a response that is not only accurate but also nuanced and detailed.

The modularity of the process, particularly with the use of Elasticsearch and separate pipelines for embedding and retrieval, ensures that the system can handle large volumes of data and queries efficiently. The object-oriented implementation allows us to adapt each individual processing step with ease and replace used methods and models at a later time to ensure a state-of-the-art solution. For example, instead of using the cloud-based GPT-4o mini model, on premise a Lama3 model can be employed quickly to ensure that sensitive webcontents and documents are processed within the company.

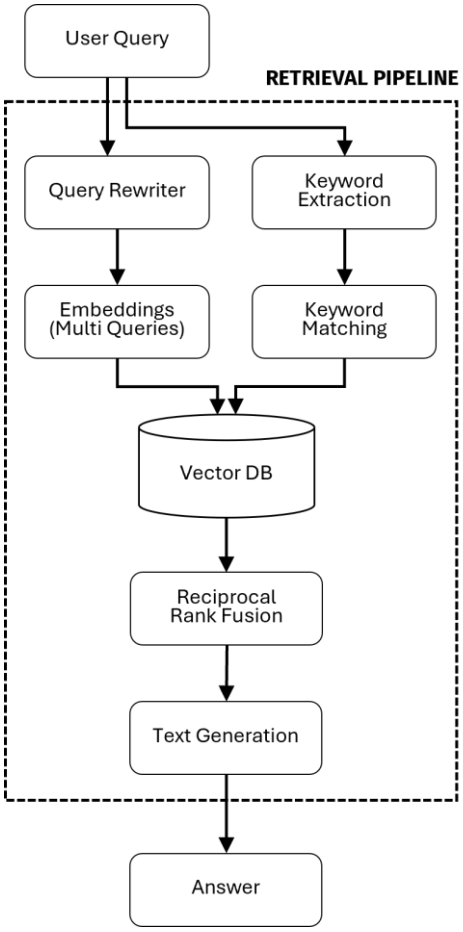

Figure 2: Retrieval Pipeline

Technology Stack

The used technology stack integrates various programming languages, databases, and machine learning

models to create a robust system for handling document metadata, content retrieval, and automated answer generation.

(1) Programming Languages: (a) Python 3.10 is the latest version of Python, known for its simplicity and readability. It is used in conjunction with FastAPI, a modern web framework for building APIs with Python. FastAPI is chosen for its speed, ease of use and support for asynchronous programming, making it suitable for high-performance applications.

(2) Databases: (a) MySQL is a widely used relational database management system. In this setup, MySQL 8 is employed to store document metadata, which includes details like document titles, authors, dates, and other relevant information. (b) Elasticsearch is a highly scalable open-source search and analytics engine. Version 8.8 is used as a vector database in this stack, storing and querying document snippets along with their embeddings. It enables quick retrieval of relevant document snippets.

(3) Machine Learning Models: (a) Text-embedding-3-large: This model is used to convert text into high-dimensional vectors (embeddings) that capture the semantic information of the indexed content. (b) GPT-4o mini: GPT-4o mini is utilized to extract keywords from documents, which can help in summarizing content, tagging, and improving search relevance. It is also used for generating coherent and contextually appropriate final answers. It takes the output from previous components (e.g. relevant document snippets and keywords) and crafts detailed, context-aware answers.

Conclusion

We have presented a Liferay plugin specifically designed to enhance user interaction and information retrieval. The architecture is built on a microservices framework, utilizing Liferay 7.4 for the user interface and FastAPI for backend services. Elasticsearch serves as a vector database, facilitating semantic search and efficient document management. The clear separation of tasks ensures a scalable and maintainable solution, while the use of modern technologies and advanced machine learning models provides a powerful and intelligent interaction with the stored webcontents and documents.

From an operational perspective our plugin offers several key benefits: it significantly improves user engagement by providing instant, accurate responses, delivers timely and relevant information without the need for manual searches, and reduces the workload on support teams by handling common questions efficiently. This allows support staff to focus on more complex issues, optimizing resource allocation and improving overall efficiency.

References

[1] Dodgson, J., Lin, N., Peh, J., Rafhael, A., Pattirane, J. Alhajir, A.D., Dinartho, E.D., Lim, J. and Ahmad, S.D. (2023) Establishing Performance Baselines in Fine-Tuning, Retrieval-Augmented Generation and Soft-Prompting for Non-Specialist LLM Users

[2] Chen, J., Lin, H., Han, X. and Sun, L. (2024) Benchmarking Large Language Models in Retrieval-Augmented Generation, in: Proceedings of the 38th AAAI Conference on Artificial Intelligence, p. 17754-17762

[3] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukin V., Goyal, N., Küttler, H. Lewis, H., Yih, W., Rocktäschel, T., Riedel, S. and Kiela, D. (2020) Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, in: Proceedings of the 34th International Conference on Neural Information Processing, p. 9459-9474

[4] Topsakal, O. and Akinci, T. C. (2023) Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast, in: Proceedings of the International Conference on Applied Engineering and Natural Sciences, p. 1050-1056

[5] Nakhod, O. (2023) Using Retrieval-Augmented Generation to Elevate Low-Code Developer Skills, in: Artificial Intelligence, No. 3, p. 126-130