# Development of an End-to-End Web Application Leveraging Retrieval Augmented Generation (RAG) and OpenAI's API with Enterprise Data

**Course**: Generative AI

**University**: Dauphine-PSL University

## Project Description

This project aims to develop a robust web application that integrates the principles of Retrieval Augmented Generation (RAG) and OpenAI's API. The application will utilize enterprise data to demonstrate the potential of Generative AI in enhancing productivity and task automatization, especially in customer interaction and customer service.

## Datasets

Simulated enterprise data will be provided, which includes:

* Twitter’s customer support data: tweets from customers and replies from customer service
* ~~Customer’s emails and customer service agent’s responses~~
* ~~Customer service call transcripts~~
* ~~Company Knowledge base PDF document~~
* ~~Company’s FAQ~~

## Objectives

1. Product Brainstorming:

* Analysis and understanding of the available data.
* Identifying opportunities where Generative AI can provide value.
* Exploration of potential applications using RAG and Generative AI on those data.
* Choose an idea of web application to implement.

2. Data Retrieval and Formatting:

* Extracting, cleaning, and organizing the provided data.
* Formatting the data to be compatible with a RAG system.

3. Development:

* Architectural design of the web application.
* Building a functional back-end components of the application using Flask in Python, and integrating Azure OpenAI's API and a RAG system.
* Building a user-friendly front-end using HTML, CSS and Javascript.
* Store data in a vector database, such as ChromaDB.
* Version control with Git.

4. Deployment:

* Deploying the application on an appropriate web platform.

## Expected Deliverable

* A fully operational web application that demonstrates the practical application of Retrieval Augmented Generation and generative AI using enterprise data.
* The project is available on GitHub.

Project Timeline

The project’s Github need to be sent before 15/03/2024.

## Advice

Begin by comprehending the data and establishing a clear objective for your application. Initially, construct a basic prototype of the front-end interface, utilizing ChatGPT for this purpose. Once the primary features of the front-end are operational, proceed to develop a straightforward Flask back-end. This could start as simply as returning the input message. Ensure the Flask server is operational and effectively communicating with the front-end.

Next, enhance the Flask back-end by integrating the Retrieval-Augmented Generation (RAG) system. Utilize the provided sample file to enable the system to identify and respond with the most relevant tweet, using GPT-generated answers. Initially, employ an Excel file for storing embeddings, progressing later to a more sophisticated vector database solution, such as ChromaDB.

Further refine the application by improving the prompts, retrieval methods, and front-end features, such as adding translation and text reformulation capabilities.

Finally, focus on deploying your web application. For hosting the Flask application, consider using a free platform like PythonAnywhere, which allows for hosting, running, and coding Python in a cloud environment.

## Resources

1. Github of the Translation App for the code structure: [End2EndAI/travel-ai-translator: This repository is a template for anyone wishing to build quickly a web application using OpenAI technologies, such as GPT or Whisper. You are welcome to use the code template for your own projects! (github.com)](https://github.com/End2EndAI/travel-ai-translator)
2. Guide to build the RAG system: [Guide from OpenAI to build a Q&A Chatbot using OpenAI API (back-end)](https://platform.openai.com/docs/tutorials/web-qa-embeddings)
3. Guide to use the Microsoft Azure OpenAI API
   1. [GPT model](https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/chatgpt?tabs=python&pivots=programming-language-chat-completions)
   2. [Embedding model](https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/embeddings?tabs=python)
4. Guide to implement a ChromaDB vector database: [🔑 Getting Started | Chroma (trychroma.com)](https://docs.trychroma.com/getting-started)
5. Free hosting for Flask app: [Host, run, and code Python in the cloud: PythonAnywhere (www.pythonanywhere.com)](https://www.pythonanywhere.com/)
6. How to use Git and how to push on GitHub: Ask ChatGPT 😉