Practice Project Overview

For the practice project, you will design a food recommendation system from a collection of almost five thousand different food items. Your system will provide recommendations from the collection based on user queries.

Practice and final project similarities

You will notice that the practice project parallels the final project, except that in the final project, your system will recommend jobs based on user queries. We strongly encourage you to complete the practice project first. We will provide you with step-by-step instructions and code to complete it. For the final project, which you will submit for grading, you will need to develop the system independently.

Practice project description

The system will use natural language processing models so it can take inputs such as

- dairy-free Greek vegetarian
- gluten-free alternative similar to pound cake
- vegan dessert without nuts

Based on these queries, your system will identify foods that meet the appropriate requirements and provide recommendations. You will also be able to order the recommendations, providing the closest matching items to the user's query first.

Also, perhaps your user has a favorite recipe in a PDF for an entrée and wants to find other similar-tasting foods. Your food recommendation system will also take a PDF as input then search and provide results based on the ingredient data provided in a PDF format.

Technologies used

The instructions for the practice project explain how to create this system by using the ChromaDB vector database and packages from Hugging Face to generate vector embeddings and identify the food items your users search for. Note that in the final project, you may choose whichever database that handles vector data you prefer, such as Cassandra, Postgres, or MongoDB.

We will provide you with a diverse array of food data. You will identify appropriate recommendations from your data using similarity search libraries and some Hugging Face natural language processing (NLP) models. By the end of your project, your database will adeptly match users with their culinary preferences.

Let's take a closer look at the two parts of this project, the text-based search and the PDF-based search.

Part 1 breakdown

In part 1, you generate a food recommendation system, which allows users to search your database using text and provide the top five ranked results. You will start by setting up the environment, installing *ChromaDB*, and creating a *ChromaDB* instance. You will also need to create a collection from a given data file.

Additionally, you will need to write code to accomplish the following:

1. Create the collection
2. Convert the collection data into vector embeddings
3. Store those embeddings in your database
4. Convert the user input into a vector embedding
5. Compare the embeddings in the database to the converted user input
6. Rank and return the top five matches

More specifically, you will be provided with step-by-step instructions for writing the JavaScript functions to accomplish these tasks. Functions you will create include:

Function or variable name	Purpose
getOrCreateCollection()	create the collection from the data file
foodItems	An array that stores the data from the data file
uniqueIds	A Set() object which stores the identifiers for items stored in foodItems
foodItems.forEach()	create unique IDs for items in the collection and store the IDs in a new Set() object named uniqueIds
uniqueIds.add()	Adds each food's ID to the set object
foodItems.map()	Combines each food item's name, description, and ingredients into a single string
foodTexts	Stores each food item's string as an element in an array
generateEmbeddings()	Generates the embeddings from the foodTexts array
classifyText()	Helper function that classifies the query into one or more labels using the BART large language model
extractFilterCriteria()	An asynchronous function that calculates how closely related the query is to the provided criteria, called labels.
filterCriteria	Variable that stores the result from extractFilterCriteria
performSimilaritySearch()	Calculates the cosine similarity between the query and the stored embeddings and rank the top five results
main()	Calls the other functions to perform the search

Part 2 breakdown

In part 2, you generate a similar food recommendation system, but this one allows users to search your database using a PDF rather than text. The setup of this system is different from part 1, although the systems behave similarly.

Function or variable name	Purpose
foodItems	An array that stores the data from the data file
extractTextFromPDF	Stores the contents of the PDF in a text buffer from the results of an anonymous function which replaces newline characters and multiple consecutive spaces with a single space to normalize text formatting
generateEmbeddings()	Generates the embeddings from the extracted text using the Hugging Face inference model
promptUserInput()	Prompts the user for the path to the PDF
extractIngredients()	Extracts the ingredients from the PDF file
storeEmbeddingsInChromaDB()	Stores the embeddings into the Chroma DB instance.
main()	Calls the other function to perform the search
collection	A variable that stores the collection data retrieved from Chroma DB
getCollection()	A Chroma DB API method which retrieves the collection from the database
results	Stores the top five vector embedding matches after comparing the collection to the recipe ingredients

We will provide you with step-by-step instructions for writing this code in the practice project.