Introduction

Food is one of the most fundamental aspects of human life, not only for survival but also for personal enjoyment and cultural expression. While dining out offers convenience, many people prefer cooking at home to explore their culinary creativity, save money, or customize dishes to suit their personal tastes and dietary needs. A typical behavior in terms of people making food is that people tend to look up food recipes online to make the cuisine that fulfills their taste. Recognizing this widespread behavior and the growing demand for accessible culinary resources, our team identified an opportunity to address this need through technology. However, many cooking websites require either a login or payment in order to view the food recipes, which we think is not affordable for certain groups of people. Therefore, we developed a project aimed at enhancing the cooking experience by designing and implementing a comprehensive solution for recipe discovery and recommendation.

At the heart of this project is an extensive database created by crawling a popular cooking website, amassing thousands of diverse recipes that cater to various tastes, diets, and cultural preferences. This dataset is rich in unstructured text data, encompassing diverse culinary options that cater to a wide range of tastes, diets, and cultural preferences. Leveraging this rich data index, we designed and implemented a search engine that empowers users to find recipes effortlessly by entering relevant keywords, such as ingredients or dish names.

To further elevate the user experience, we integrated a recommendation system that analyzes past search patterns to suggest recipes tailored to their unique tastes. By combining advanced search capabilities with personalized recommendations, our project not only streamlines recipe discovery but also encourages culinary exploration, making cooking at home more accessible, enjoyable, and rewarding for everyone.

A key aspect of the project is the logging mechanism, which records all user interactions to enhance personalization and improve keyword recommendations with the increasing number of searches. This functionality is implemented by logging the search keyword into a simple text file for each search.

Design Details

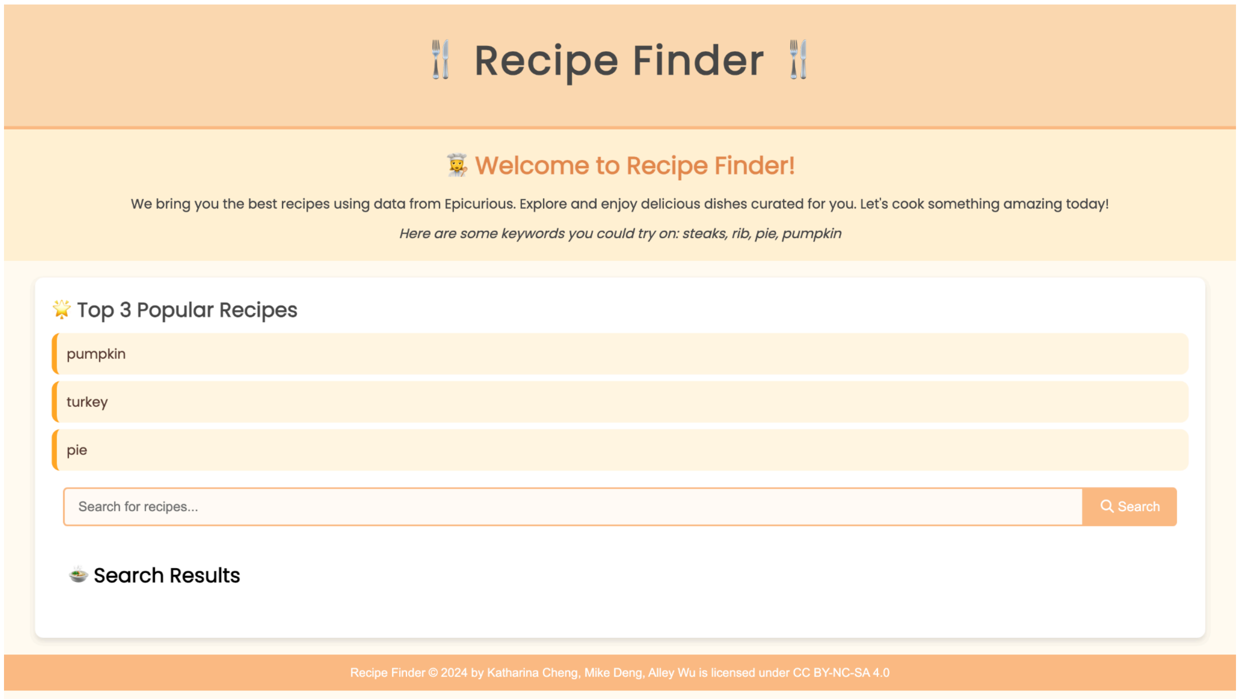
We used PyTerrier to index our recipe documents. To begin, we chose pt.FilesIndexer() over pt.TRECCollectionIndexer() to configure the indexer, as our input documents are in HTML format. The pt.TRECCollectionIndexer() function only accepts TREC-formatted collections, which is incompatible with our dataset. Our recipe documents were collected from a website called Epicurious. If our collection were in TREC's dataset format, we would have opted for pt.TRECCollectionIndexer() instead. After selecting the appropriate indexer function, we defined metadata fields and their corresponding lengths. The fields we configured were docno, a unique identifier for the document with a length of 26 characters, title, the document's title with a length of 256 characters, and description, an introduction to the recipe with a length of 512 characters. Once the metadata fields were defined, we mapped them to the appropriate document tags in our dataset. With the configuration complete, we indexed all the files in our dataset and stored the result in a variable called indexref. After indexing, we used pt.IndexFactory.of(indexref) to access the created index for querying. Finally, we retrieved and reviewed key statistics from the indexed data using getCollectionStatistics().toString(). The statistics showed that 927 documents were indexed, 70,625 unique terms were identified, there were 4,055,221 postings, and 71,765,779 tokens in total.

The number of documents we indexed fell slightly short of our initial goal of 1,000 documents. This shortfall was primarily due to the limitations and variability of crawling the Epicurious website in a fixed amount of time using the wget -r command. Factors such as network conditions and server response times resulted in a variable number of documents collected per run. Rerunning the command could result in more or fewer documents being collected, so we did not recrawl the Epicurious website when we collected 927 documents, which was very close to the 1,000 documents.

The retrieval model we used in this project was Best Match 25 (BM25). We selected BM25 because we believe it better captures user intent compared to other models, such as TF-IDF or Language Models (LM) when evaluating the relevance of recipe documents. There are two main reasons for this choice. First, BM25 prevents repeated terms from inflating a document's relevance score, which is particularly important for avoiding bias toward recipes with repetitive wording. Second, our recipe collection includes documents of varying lengths, and BM25 excels at balancing the relevance of short and long documents. This ensures that longer recipes don't automatically rank higher simply because they contain more terms. BM25 provides fairness by allowing shorter but highly relevant recipes to compete effectively with longer, less relevant ones. For these reasons, we chose BM25 as the model to evaluate and rank the relevance of our recipe documents.

We did not apply stemming or stopword removal during indexing. After comparing the BM25 evaluation results for documents with and without these preprocessing methods, we found that the documents without stemming and stopword removal achieved higher BM25 scores. This outcome can be explained by the nature of recipe documents. Some words classified as useless words are important in the context of recipes. For example, in "orange with chicken," stopword removal could reduce the phrase to "orange chicken," which alters its meaning and makes it harder for BM25 to match queries accurately. Similarly, stemming can overgeneralize terms. For instance, "baking sweet potato" and "baked sweet potato" carry different meanings in recipes. However, stemming would reduce both to "bake sweet potato," which could make it harder for BM25 to accurately match user queries, such as when a user is specifically searching for recipes involving cooked sweet potatoes rather than raw ones. For these reasons, we chose not to use stemming or stopword removal, ensuring that the BM25 model could better preserve the context and meaning of our recipe documents.

Usage scenario


🍴 Recipe Finder 🍴

🧑‍🍳 Welcome to Recipe Finder!

We bring you the best recipes using data from Epicurious. Explore and enjoy delicious dishes curated for you. Let's cook something amazing today!

Here are some keywords you could try on: steaks, rib, pie, pumpkin

🌟 Top 3 Popular Recipes

pumpkin

turkey

pie

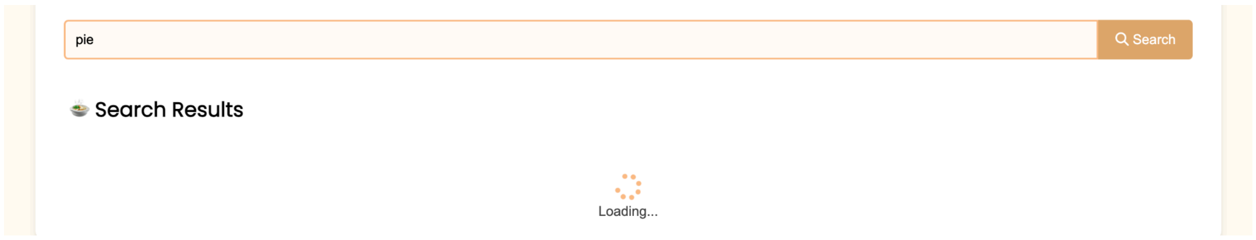
🍲 Search Results

Recipe Finder © 2024 by Katharina Cheng, Mike Deng, Alley Wu is licensed under CC BY-NC-SA 4.0

The picture at the top is what the website looks like. At the top of the website, we feature the title "Recipe Finder" followed by a brief description of the website's purpose. Users are also provided with four example queries to get started: steaks, rib, pie, and pumpkin. The website consists of three primary features:

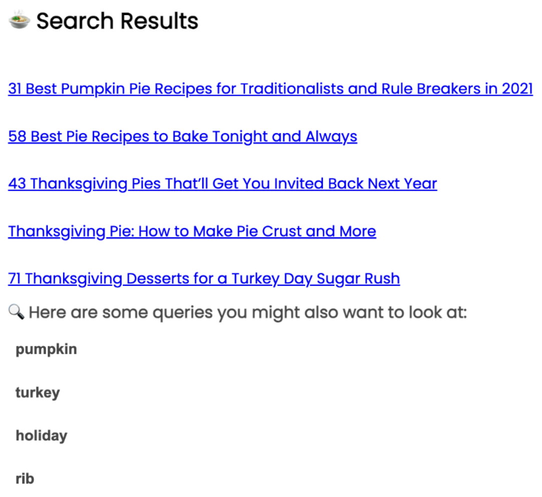
1. Top 3 Popular Recipes
2. Search Results
3. Mapping Recommendations (a hidden feature)

Additionally, a Creative Commons license is included in the footer to attribute and license the project appropriately.



The three most popular search terms are dynamically identified and displayed via the Top 3 Popular Recipes feature. Exploring previously trending queries is made easier by these well-liked recipes appearing as clickable items that users can select to instantly populate the search bar. This capability is made possible by an effective logging and analyzing procedure. The file_put_contents() function is used to append each user-inputted search query to the recipe_search_logs.txt log file. Unix programs like uniq -c for counting and sort -nr for ranking are then used to process the logged searches, and the output is saved in top_queries.txt. This guarantees that the most frequently asked questions are updated continuously and shown in decreasing frequency order.

The Recipe Finder website's core feature is its search capability. A visual loading spinner and a real-time search process are both triggered when users enter search words into a search box. By giving consumers feedback, this spinner makes sure they are aware of the system's current situation. The search query is recorded in the recipe_search_logs.txt file on the backend, and retrieval based on the BM25 ranking model is carried out using the PyTerrier module. To provide users with precise and insightful recipe recommendations, this model determines how relevant search results are to the input query. For ease of browsing, the results are then shown as clickable links.



Search Results

31 Best Pumpkin Pie Recipes for Traditionalists and Rule Breakers in 2021

58 Best Pie Recipes to Bake Tonight and Always

43 Thanksgiving Pies That'll Get You Invited Back Next Year

Thanksgiving Pie: How to Make Pie Crust and More

71 Thanksgiving Desserts for a Turkey Day Sugar Rush

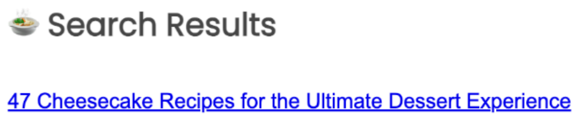
🔍 Here are some queries you might also want to look at:

pumpkin

turkey

holiday

rib



Search Results

47 Cheesecake Recipes for the Ultimate Dessert Experience

A unique feature of the website is its Mapping Recommendations system, which operates behind the scenes to analyze query patterns. A mapping of user activity is produced by pairing each query a user logs with a subsequent search. The current query is represented by the key in this layered structure, and the value is another dictionary that contains the upcoming requests and their frequencies. The information is utilized to generate suggestions and is stored in related_queries.json. If a user searches for "pie," for example, the system may recommend related terms like "pumpkin" based on past user activity. If a query has no associated patterns, the "Here are some queries you might also want to look at" section remains hidden, ensuring the interface remains clean and professional.

Known issues and future work

During the development of this project, we encountered several challenges and identified areas for future improvement.

1. Difficulty Finding a Suitable Recipe Website

Finding a recipe website that could be crawled and indexed was one of the first problems. Our ability to gather a huge dataset for indexing is complicated by the fact that many well-known websites either block automated scraping or offer APIs with restricted access.

2. Indexing Valid Documents

Issue: PyTerrier was unable to recognize valid documents during the indexing process because the files lacked the appropriate .html suffix, resulting in skipped or ignored documents.

What we have done: A Linux script was used to add the .html suffix to all files in the document folder, ensuring they were recognized as valid HTML documents: *for file in *; do mv "$file" "${file}.html"; done* This allowed PyTerrier to index the documents successfully.

Future Work: Automate the file renaming process within the crawling pipeline to avoid manual interventions and ensure compatibility with indexing tools.

3. Displaying Search Results

Issue: The search results container's original version contained lengthy, repetitive, and irrelevant phrases for some search terms, which made the user interface crowded and could have confused users.

What we have done: In order to guarantee that only distinct and significant search results were shown, we modified the rendering logic to eliminate duplicate containers and filtered the PHP code to eliminate irrelevant phrases.

Future Work: To improve the user experience, add pagination to the search results and filtering options in the user interface.

4. Irrelevant or Empty Search Results

Issue: Results from certain queries weren't very pertinent to what the user was looking for. Queries that were not indexed in the dataset produced blank answers that gave the user no feedback.

Future Work: To increase the relevancy of search results, we ought to think about incorporating a query-expansion mechanism or a semantic search model for the irrelevant ones. To better comprehend user intent, this may entail using Large Language Model (LLM) or Natural Language Processing (NLP) approaches. Think about including a fallback option that either displays a notice like "No exact results found" or proposes other inquiries if a query returns empty results. Try looking up relevant terms online or perusing well-liked recipes. To increase user satisfaction and direct users toward relevant queries if no results are obtained, we may implement a related-query suggestion system.

5. Missing Recipe Ratings

Issue: We were unable to successfully scrape the ratings that were available on the original website, so we were unable to use collaborative filtering approaches for the recommendation system without ratings.

Future Work: Create a more sophisticated scraping technique to extract ratings or look at other methods, such as the frequency of references in user reviews or comments, to determine how popular a dish is.

6. Selection of BM25 for Document Ranking

Issue: Because of its ability to manage document length variations—a crucial consideration for our recipe dataset, which includes lengths ranging from brief instructions to comprehensive guides—BM25 was selected. Longer recipes or ones with a lot of terms could dominate rankings without normalization, producing less relevant results.

Future Work: Examine Advanced Models: For improved semantic comprehension, look into transformer-based models (like BERT). Additionally, we may combine BM25 with variables like ratings or the popularity of the recipe.