

UNIVERSITY OF EDINBURGH

GROUP PROJECT

My Perfect Recipe

Authors:

Yashvi Chawla
Rajasekhar Malireddy
Jiale Peng
Anna Ivahnenko
Ting Sheng Tan
Chentao Yang

February 28, 2020



Abstract

The search for the recipe that a person has in mind is an easy one, whilst the search for the recipe that is possible to make given the ingredients available is challenging. The idea behind this project was to implement a recipe search engine that provides the recipe with minimum extra ingredients needed to buy and the least time required to prepare. The recipe suggestion also takes account of the weather in the user's location, special calendar events and standard diets. This search engine required novel ways of preprocessing and ranking of 102K+ recipes provided on by five of the most popular English language cooking sites.

Method and Tools Used

The following sections describe our thought process, rationale and the tools we had to learn and used in by doing this project.

Crawling

To determine our limitations for this project we first designed a full-fledged crawler with the capabilities of crawling the entire Internet using the techniques of multi-threading and locks. Choosing to focus on a specific everyday problem solution which we believe could be solved with text technologies, the way in which we collect and process webpages had to be re-designed. Our aim narrowed to focus on a more specific domain of recipes where we, first of all, explored the different site formats and noted the content that was highlighted such as the number of servings, cook time, prep time, rating etc.

As the structure of many of these sites varied greatly we chose to focus on the five largest and most prominent English language food sites, which where: 'bbc.co.uk/food', 'allrecipes.co.uk', 'www.epicurious.com', 'www.food.com', 'myrecipes.com'. It took patience to understand the subtle nuances of each page such as URL differentiations as well as HTML and JavaScript elements. Even within these five sites the rating elements were inconsistent and in the end due to the short time scale, we chose not to consider them.

In our implementation, the crawler was able to crawl the websites completely in one go, where the threads worked concurrently using shared memory. On a local computer, we used 10 threads, while later on Google Cloud we switched to 16 threads for better efficiency. We used a robot file parser library to access the robots.txt file of every website we crawled so as to ensure a fair policy of web crawling. The Robots.txt file was able to tell us if we could crawl the website page or not. We used timestamps to ensure that threads waited for the right amount of time as specified by the robots.txt file before they could crawl the page again. Our crawler was efficient enough to handle redundancies and was able to omit recursive crawling.

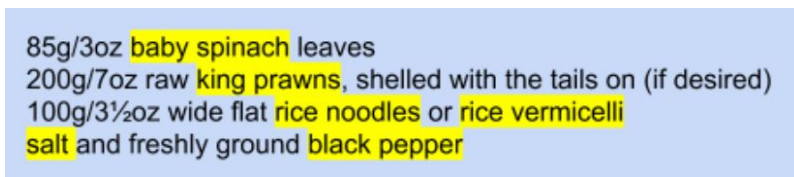
Throughout this process we had to make crucial design decisions, making sure everyone is involved, communicating our issues while listening to the team's opinion and needs. Through this process, we understood that for preprocessing the crawler output needs to have clearly defined structure, while the front end design needed to consider how the information collected had to be presented to make the product intuitive and easy to use, we accommodated this by including image URLs. Therefore the chosen output file structure was broken down into 8 parts: (1) recipe URL, (2) image URL, (3) recipe title, (4) description, (5) preparation time, (6) cooking time, (7) multiple ingredient lines and (8) end tag to signify the start of a new recipe.

Preprocessing

In this stage, all the crawled recipe content files are processed to then be stored into a MySQL database. Firstly, each recipe content file was read and then processed line by line to extract the recipe content, made easy by the structured format of the web crawl content file. When parsing an increment counter was used to track the number of recipes that have been processed and also used as a unique recipe ID in order to distinguish between different recipes later. Then, recipe web URL, recipe image web URL, recipe title, recipe description, preparation time, cook time, and serving count were extracted line by line from the recipe content and stored tentatively in Python variables.

To classify recipes into cuisines (e.g. Scottish, American), diets (e.g. vegetarian, dairy-free) and special events (e.g. Christmas, pancake day) the title and description of the recipe was used. A simple boolean search was implemented for 70 selected classes. This produced robust results since almost always the nature of the recipe was stated in its description. In future work, this feature can be extended by using machine learning to classify recipes by resulting clusters and the results used for further query expansion.

The ingredient section of each recipe content consisted of a variable-length list of ingredients. For each ingredient, the description was presented in a separate line in the recipe content file, identical to how the information was shown on site. However, parsing this data was a huge challenge in the preprocessing stage because each description could vary in structure, measures, descriptive adjectives, cooking techniques or equipment. While ingredients could range from a short 3-5 words or contain multiple ingredients in a single line. Just some of these issues are illustrated in Fig. 1.



85g/3oz baby spinach leaves
200g/7oz raw king prawns, shelled with the tails on (if desired)
100g/3½oz wide flat rice noodles or rice vermicelli
salt and freshly ground black pepper

Figure 1: Example of ingredient description, illustrating the challenges for preprocessing.

The ingredient preprocessing part was divided into three parts. First looking for keywords: ‘and’, ‘or’, ‘and/or’, ‘plus’, ‘plus extra’. This is done to make sure each part is processed correctly and the ingredient count for the recipe is correct in each instance. So in the case of ‘and’ and ‘plus’ an extra ingredient was added to the list and the ingredient count increased, in contrast to ‘or’ and ‘and/or’ where the extra ingredient was added to the list but the count did not increase. While for the ‘plus extra’ case the following was discarded since it usually contained a description as to why.

The second part looked for common and easily confused phrases (e.g. ‘chicken’, ‘oyster mushroom’, ‘black pepper’) within each ingredient line so that even if they are mentioned they could be classified straight away and are not removed in stage 3. This also helped solve the problem of multiple ingredients in a single line and classify the ingredients into broader categories that could be used for query expansion (e.g. ‘beef mince’ also stored under ‘beef’).

While the third part discards the text description in brackets and after the first comma which almost always contained a description. Afterwards, the individual line was tokenised, case folded and passes thorough uses a specifically designed extensive stop words list to produce a clean list of ingredients, containing categories, common and uncommon ingredients (e.g. ‘oil’, ‘olive oil’, ‘free-range’, ‘egg’, ‘free-range’, ‘egg’, ‘bratwurst’). This clean list is then Porter-stemmed and stored in a Python variable.

Query Expansion

Creating a clean list of ingredients also allowed us to build in query expansion by constructing a lookup table of synonyms ingredients (e.g. ‘green onion’ mapped to ‘scallion’ and vice versa) and map groups of ingredients together for dairy, meat, grain etc. This was done manually for most of the ingredients present (22K). In future work, we can use machine learning to class new ingredients and work on ways in which we can be sure that groups of ingredients are excluded if the user searches for a specific dietary requirement.

Indexing

To make the search efficient an inverted ingredient-based index mapping ingredients to recipe IDs was created. For our posed problem ingredient the index can be one or more words e.g. ‘chicken’, ‘passion fruit’, and ‘cheddar cheese soup’ are all valid entries. Besides the inverted index, recipe information MySQL table is also created storing the URL, image URL, the time needed and the servings and the number of ingredients needed by the recipe.

To construct and communicate with MySQL tables, a ‘Database’ Python class that contains back-end functions was written. This class was used to automatically create the Ingredients Index and the Recipe Information tables and allowed the data to be added as each recipe is processed. A method was also written for retrieving the ingredients and recipe info from the database. More information can be found in the appendix.

Additionally, to store our inverted index table more efficiently, we implement Delta Encoding and Variable Byte Encoding. Instead of storing a list of recipe IDs into the inverted index, we stored the differences between recipe IDs. Then, we use Variable Byte Encoding to represent those differences in order to further reduce the size needed to store those recipe IDs in our inverted index table.

In a nutshell, following is the schema of the tables in our database.

Celebration_calender schema

|Date|Celebration|Message|

Ingredient Index table schema

|Ingredient|RecipeID(inencryptedformatandforeignkeystorecipeinfo|table)

Recipe Info table schema

|RecipeId|Recipeurl|ImageUrl|title|description|preparationtime|cooktime|servingcount|Ingredientcount|

Synonym table schema

|ingredient|synonym(foreignkeytotableingredientindex)|

Ranking

This project presented a novel problem that required a customized search algorithm to serve a unique purpose. The solution was partly driven by user design features in the front end such as the search bar, checkboxes and search options. The first feature allowed the user to search for the main ingredients of their choice, while the drop-down checkboxes listed common secondary ingredients that the user already has but does not have to type. The third feature showed the user the options ‘Very Hungry’ and ‘I can go shopping’ implemented as checkboxes, by adding these we could find if we had to prioritise by time and/or the number of ingredients needed to be brought.

To find the number of ingredients the user needs to buy our algorithm subtracted the number of ingredients users already has (including the ingredients from the search box) from the total number of ingredients of corresponding recipe results. This trick allows us to create a dictionary with recipe IDs as keys and the number of ingredients required to buy as values.

The algorithm covers 4 cases which mainly revolves around two parameters, time required to cook and the number of ingredients user has to buy before they can cook:

1. User is *very hungry* and *can go shopping*

As the user is hungry so they are probably looking for recipes which take less time to cook, we have given importance to time than to the minimum no of ingredients to buy as the user is willing to go shopping.

2. User is *very hungry* and *cannot go shopping*

More importance is placed on finding the minimum number of ingredients to buy with the highest priority given to recipes with 0 ingredients to buy. Moreover, these recipes are sorted by time so that the search would return recipes with less time required at the top.

3. User is *not hungry* and *cannot go shopping*

The algorithm prioritises results with 0 ingredients to buy as user does not want to go shopping.

4. User is *not hungry* and *can go shopping*

User is not hungry and can go shopping Importance is then simply given to the minimum number of ingredients to buy.

Retrieving

When landing on the home page the initial results show recipes corresponding to either one of the 47 specific calendar events (such as Diwali or Thanksgiving), if it falls on that date. Otherwise, recipes pertaining to the weather of the user’s location are shown. For instance, if the weather is cold ($<10^{\circ}\text{C}$), then the recipes of hot nature will be shown.

The location of the user is retrieved using an IP address from the user request packet and ‘pygeoip library’ which maps the IP address to the location of the user via geolite.dat file. The latitude and the longitude parameters are used to retrieve the weather status such as temperature, pressure, humidity etc. from the ‘Darksky’ (Weather Website) API. The current weather status (windy, cloudy, rainy, clear sky and snowy) and the time from the user browser (morning, afternoon and evening) are used to display a message to the user. As of now, the user has no discretion in the way our website accesses the user location and IP address.

The basic retrieval process mirrors the main steps taken during preprocessing. First tokenizing the ingredients from the query box by punctuation and whitespace and then lower casing them. Since the ingredients might consist of more than two words, permutations of all listed words are considered during the search. For each permutation, the synonym table is accessed to retrieve the synonyms and query is further expanded with these synonyms along with the permutations. The ingredient index table is then accessed to retrieve the recipe IDs based on all query permutations and synonyms. As the recipe IDs are encrypted using Delta and Variable Byte Encoders (mentioned above) for efficient compression, we decode the recipe IDs using Delta and Variable Byte Decoder to get a final list of recipe IDs.

Our system is capable of handling network errors and other exceptions. The user has the option to input their query in any given format. One future improvement would be to build in query completion and spell checking, however, most common mistakes such as ‘yogurt’ or ‘yoghurt’ will be covered by the query expansion synonyms list.

Connecting to Front and Back End

We chose to use the name ‘MyPerfectRecipe.co.uk’ for our website as it reflects personalisation aspects and our goal of providing users with a recipe that exactly suits their needs. This domain was registered through GoDaddy, a major domain registrar.

The backend was written using Flask Framework (Python) to connect to the frontend and catch user requests. PythonAnywhere online web hosting service was used to host our website because it supports easy deployment for Flask code.

To provide functionality, 3 major functions with many other supporting functions were written to process the user request and retrieve recipe IDs and content: 1.getip(), 2.fetchdata() and 3.retrievedata(). The request flow is divided into a 2-way communication process to achieve better efficiency in terms of computation and memory. Even though it is possible to retrieve results and display in the frontend in one go, it would be computationally expensive to send a large number of recipe data from the backend to the frontend and thus we resort to a 2-way communication process to handle the user request. During this process, the frontend can store the recipe IDs in memory so for additional results we have to connect to the database only once.

During the front-to-back docking process, we adopted AJAX (‘Asynchronous Javascript And XML’) technology to transfer parameters. In order to achieve AJAX interaction, we need to install the Axios module in the programming environment. In addition, during the front-end and back-end interactions, parameters are passed in the form of a dictionary, so as long as we unify the keys of the dictionary, we can correctly implement data interaction

When the user lands on the home page, function mounted() in JavaScript (JS) calls the function getip() in the backend with the user browser time as a parameter. getip() checks for the occasions by accessing the ‘celebration’ table in SQL, give back the recipe IDs by accessing the ‘ingredient_index’ table to the frontend. If no occasion is occurring, getip() checks for the weather status at the user location and returns the pertaining recipe IDs to the frontend. Then mounted() calls the function retrievedata() function in JS function calls the retrieve() function in backend by sending 9 recipes IDs as parameters. Function retrieve() access the ‘recipe_info’ table and retrieves the content using recipe id as a key. The function retrievedata() also keeps track of the counter of the master recipe IDs array so that it would not send duplicates that have already been fetched from the backend. In this manner, we had mitigated the network traffic and increased efficiency.

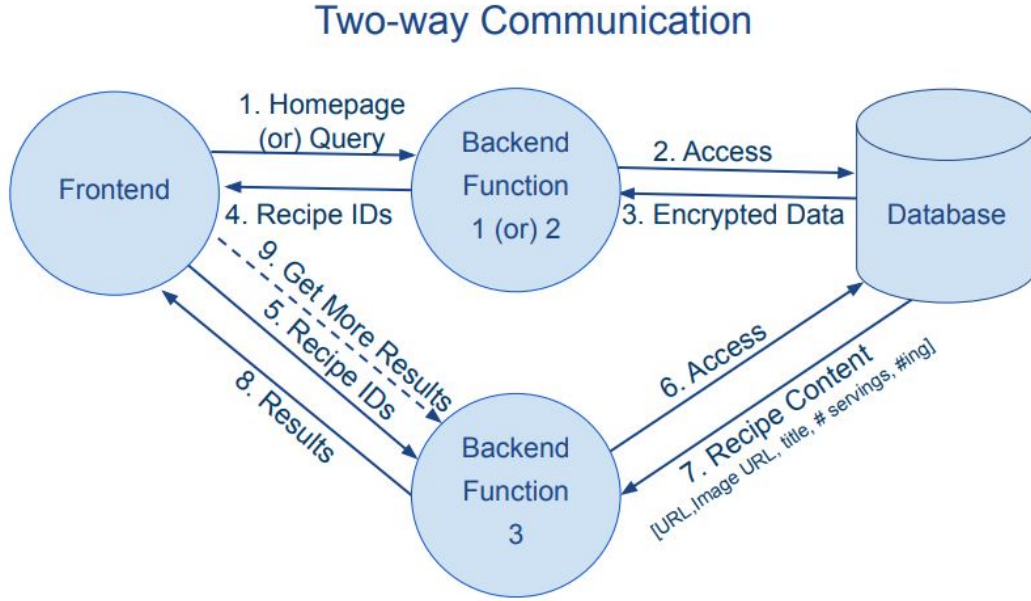


Figure 2: 2-way communication process

When the user types the ingredients and clicks the search button, `handlesearchclick()` function in JS calls other JS function `getdata()` which in turns calls the function `fetchdata()` in the backend by passing the query, checkbox values to the backend as parameters. `fetchdata()` would retrieve the recipe id's based on the algorithm we discussed in the ranking section and gives back the results to `getdata()`. Function `getdata()` calls the `retrieveedata()` which will return 9 recipes contents from the backend.

As the website is JS enabled, images get loaded on the image grid instantly when the frontend receives the results from the backend. The button at the bottom of the page enables the user to fetch more results for the query, by calling the `retrieveedata()` function which performs the task of fetching as discussed before.

The prompt messages are loaded dynamically given situations such as no results found error, loading results message and suggestions prompt. The messages are written in such a way that the user is fully aware of what is happening and why they were given those results.

Front End

The front end design was a vital part of the project, resulting in a user-centric design that is easy and intuitive to use. From the time the user lands on the site they see the image logo that reflects the site's function and as the user scrolls down the search bar prompts the user to search and explore the checkbox options, once again showing a consistent message. Without searching the result displayed are based on the date (e.g. St Patricks or Passover) or the weather in the user's location (warm recipes on a cold day). Overall design and options were aimed to be beautifully styled and simply consisting of 2 parts the query (search bar, checkboxes, search options) and the results (results and results explanation).

Each of these sections is thought through to make it easier for the user to submit their query. Therefore the search bar is fixed and will always be shown on screen so that the user can submit their query at any

time by clicking enter or pressing the search button. The collapse checkboxes where the user can select what they already have, which can serve as a reminder of very common ingredients, easier than typing and prevents typing entry mistakes. The search options are easy to understand and access, they ensure that the results are in line with user needs.

The results section aimed to give easily understandable results. We chose to display the results pictures since this is often the best description of any recipe that the user can understand at a glance, where the title and description of the corresponding recipe will only be displayed when the user moves the mouse over the picture. The reason why those results were given is clearly stated and while loading a loading animation is displayed so that the user is able to understand what is happening at all times.

The front end was made using Vue framework, as it is a modern web development tool that allows integration of HTML, CSS and JS functions in a single file making this tool very practical and convenient. Secondly, Vue can segment the website into different parts, which can be easily edited. Using this tool we were able to set the style of each section, define JS for the transformation form of the elements and create a back-end connection.

Challenges

Finding sites with enough recipes to make a collection of the required size was challenging. Since we needed to extract particular site elements, we had to build functions that are unique for each of the five sites we crawled. Working in Google Cloud significantly improved our performance but it still took in some cases more than 24h to crawl one of these sites.

During preprocessing some of the more interesting challenges were to be able to differentiate ingredients such as ‘kidney bean’ and ‘kidney’. This was solved by using the second stage of preprocessing where each ingredient line is scanned for key phrases such as that. As well as, ingredients determine the class of ingredients such as ‘Iranian Reshteh noodles’, which had to be classified as both noodles and Iranian this was once again solved using the second stage of preprocessing.

For the problem posed, we needed to designing and defining new ranking metric and think of user needs. To determine how many ingredients the user needs to buy without searching for all the ingredients associated with it was an interesting problem that we solved by counting the total number of ingredients in the recipe and deducting the number of ingredients the user already has.

The frontend challenge was to learn a brand new framework while creating a thought-through design and providing a seamless connection to the back end. Just some of the challenges involved: Efficiently designing the backend to mitigate the network traffic and the usage of network resources using two-way communication method, retrieving weather status at the location of the user. Also when the requested results are returned, pictures are often of different sizes, needing to be formatted accordingly and when the user hovers over them the description content requested from the backend is displayed.

Future Improvements

By performing user testing we can understand more of what is expected from our search engine in terms of functionality and the types of searches made. The features listed below are some of that we think might be beneficial to explore.

- While typing the ingredients user will likely make mistakes, spell checking and query completion would be the first priority future challenges.
- We could also test how the user reacts to log-based recommendations, whether they prefer certain types of recipes or if they are adventurous with their cooking skills.
- The option to exclude a group of ingredients, that finds the dietary requirements such as ‘non-dairy’ or ‘vegetarian’ could be added as an option by using the synonyms table and ingredient groupings described. These groupings could be determined by using a clustering machine learning algorithm on either the ingredients or the recipe description and title.
- Taking the number of servings produced and the amount of each ingredient needed could also be taken into account could also be a useful feature for the user.
- Since we class the recipes into cuisines, we would be able to make recommendations based on user location.
- Be able to engage with the user if they want a dialogue (e.g. ‘What should I have today?’).

Conclusion

This is a unique idea that uses text technologies principles and produces helpful results for user’s everyday cooking needs. Optimising the results by time and/or the number of extra ingredients the user needs to buy. The user has the option to search for the recipes by the minimal time needed to prepare or the number of ingredients they will need to buy. Our approach used novel ways of preprocessing, indexing and ranking algorithm that we implemented from scratch. The user can search from over 100 thousand recipes by ingredients, diets, cuisines or calendar events to get the results that optimally match the ingredients they already have at home. Our search engine can also take into account the weather where the user is searching from and the date, to give more relevant results allowing them to plan ahead for a hot meal on a rainy day or an Easter lunch.

Appendix

Indexing Method

The following briefly explains the ‘Database’ methods:

- ‘CreateIngredientIndexTable(...)’ method is used to create an ‘ingredient_index’ MySQL table that contains four columns, which are ‘ingredient’, ‘recipe_ids’, ‘ingredient_synonyms’, and ‘ingredient_temperature’. ‘ingredient’ column contains the primary keys of the table and also acts as indexes for the inverted index. On the other hand, ‘recipe_ids’ column contains unique IDs that link to recipe information in the other MySQL table. Lastly, ‘ingredient_synonyms’ and ‘ingredient_temperature’ columns are used to support extra features of our system.
- ‘CreateRecipeInfoTable(...)’ method is used to create an ‘recipe_info’ MySQL table that contains 9 columns, which are ‘recipe_id’, ‘recipe_url’, ‘image_url’, ‘title’, ‘description’, ‘preparation_time’, ‘cook_time’, ‘serving_count’, and ‘ingredient_count’. In this table, ‘recipe_id’ column contains the primary keys to access the recipe information such as recipe web URL, recipe image URL, recipe title, recipe description, preparation time, cook time, serving count, and lastly, the total number of ingredients required by a particular recipe.
- ‘AddToIngredientIndexTable(...)’, ‘AddIngredientSynonymsToIngredientIndexTable(...)’, and ‘AddIngredientTemperatureToIngredientIndexTable(...)’ methods are used to add information into the ‘ingredient_index’ MySQL table.
- ‘GetRecipeIds(...)’, ‘GetExtendedRecipeIds(...)’, ‘GetIngredientSynonyms(...)’ and ‘GetIngredientTemperature(...)’ methods are used to retrieve information from the ‘ingredient_index’ MySQL table.
- ‘AddToRecipeInfoTable(...)’ method is used to add information into the ‘recipe_info’ MySQL table.