CMSC 122: Project Proposal
Chief Cook and Bottle Washer

1. Introduction

The idiom "*to be chief cook and bottle washer*" is one that anybody who cooks at all must be intimately familiar with; it sums up the panoply of tasks – standing in line at the store for groceries, pulling one's hair out over the dinner menu, and the oft-dreaded cleanup – that domestic cooks confront.

Our goal is to create a program that handles the first two abovementioned tasks for our users; our program will be a search engine for recipes that takes a list of ingredients the user currently has, and returns recipes found online that use those ingredients. In addition, for recipes that requires just a few more ingredients that the user lacks, our website will link them to Instacart, where they can view the prices of these ingredients, check for delivery options in their area, and purchase them.

2. Data Sources

We plan to gather data from the three recipe databases that have garnered the most visitor traffic in 2015. These databases have similarly structured web pages – a quick look at the HMTL source codes show that the ingredients sections can be easily parsed with BeautifulSoup, and the individual ingredients extracted.

With Instacart, we plan to learn techniques that will automate form-filling, so that we can populate information on amounts and ingredients onto Instacart, to search for prices and availability.

The websites are:

- allrecipes.com
- foodnetwork.com
- food.com
- instacart.com

3. Brief Plan of Work

The back-end work required for this project can be split into two distinct parts: the first is scraping the three recipe databases listed above and indexing the information on their recipes. The second involves populating user inputs on amounts and ingredients onto Instacart.

[Indexing and Web Scraping] We will have to index recipe websites. The index will be a dictionary, whose keys are single ingredients, and values are lists of urls to recipes which use those ingredients. Given user-input list of, for example, three ingredients, we return the intersection of those three values, i.e. the recipes that use all three ingredients. We could also add additional filters such as cooking style, cuisine, time taken and maybe budget for how much the user is willing to order ingredients off Instacart.

[Instacart and form-filling] We will have to figure out how to fill up forms on Instacart automatically and extract the data to be displayed to the user on a webpage. Since Instacart requires an account associated with a postal code, in order to limit the scope of our project, we will work off an individual's account, whose location is in Hyde Park, just for testing purposes.

We intend to frontload this project as our schedule at the end of the quarter is rather packed.

4. Additional Technologies/Algorithms

In order to fulfil the requirement of using new technologies, we intend to learn ***Django*** to create the user-interface of our program. Additionally, in order to automate the search process on Instacart, we will be using Python's ***Request library,*** specifically the ***POST request*** method, in order to submit our queries into the appropriate fields on Instacart's webpage.

5. Timeline and Key Deliverables

Timeline	**Objectives**	**Key Deliverable**
1/24 – 1/30 (4th Week)	Figure out detailed plan and exact methods that we will use: especially pertinent for the link to Instacart , and form-filling techniques.	Update this document with a high-level description of the program's structure and flow.
1/31 – 2/13 (5th and 6th Weeks)	Index recipes in the three target webpages. Decide on the appropriate structure for SQL tables. Create a small index for testing purposes. Using this index, write the rest of the code needed to process user's searches. Make key decisions pertaining to the way in which recipes will be returned: by relevance, or number of ingredients used, etc.	Finished index, if possible. Code that works on the smaller index. Update this document with examples of key decisions made, as part of process documentation and to justify why certain operations were used.
2/14 – 2/20 (7th Week)	If index is incomplete, continue with web scraping. Learn and use Django to create user interface. Have a working model by the of the week.	Finished index. Working Django interface

2/21 – 2/28 (8th Week)	Testing, and buffer week – we have allotted a buffer to move the schedule back by a week if necessary.	

6. Potential Issues & Solutions

Issues	**Solutions**
Recipe website might not list all available recipes via inter-connected links.	When designing web scraper, think of a way for the scraper to be able to access all target web pages. Maybe start from a page that lists recipes by sections.
User might have a ton of unrelated ingredients whose intersection is an empty set. For example, milk, chilli, cod, vinegar, chicken, beef – pretty certain that no recipe uses all of the above	Could solve by limiting our search for recipes to using only one meat product at a time, since that is typical of most recipes.
Using SQL for our index	