CAPRecipes: A Context-Aware Personalized Recipes Recommender for Healthy and Smart Living

by

Harshit Jain
B. Tech., Uttar Pradesh Technical University, India 2014

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

CAPRecipes: A Context-Aware Personalized Recipes Recommender
for Healthy and Smart Living

by

Harshit Jain
B. Tech., Uttar Pradesh Technical University, India 2014

Supervisory Committee

Dr. Hausi A. Müller, Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Hausi A. Müller, Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Departmental Member
(Department of Computer Science)

ABSTRACT

In the past few years, the general work habits of people have changed dramatically, raising concerns about their well-being. Numerous health-related problems have been observed from their health records such as obesity, diabetes or heart diseases, and unhealthy eating is one of its factors. But these problems can be prevented if people start eating healthy food. The population, in general, is also realizing that healthy eating is important for their well-being. However, they usually resist because they assume that healthy food is not tasty and they do not want to comprise their taste preferences. Moreover, they have various other considerations that become barriers for them while selecting a healthy recipe. These are:(1) their complex, restrained needs (i.e., allergies and nutritional goals), (2) their strict lifestyle or dietary preferences (i.e., their desire to eat only vegan or vegetarian food), (3) lack of knowledge about how to choose healthy recipes while exploiting their taste preferences, (4) choosing recipes that maximize the use of available ingredients in their kitchen. Numerous researchers have been working in this field and developed various applications and systems to suggest healthy recipes.

Apart from unhealthy eating, household food wastage has become a public problem, and some of the causes, which trigger it are users' taste preferences (i.e., disliking of the food), and not cooking food before ingredients expiry dates.

Thus, we propose a personalized recipes recommender system as a proof of concept called CAPRecipes, which is based on context-awareness. It tackles the aforementioned barriers and improves the users' experiences by providing the recommendations

of personalized recipes with minimal efforts while exploiting their dynamically changing contexts. CAPRecipes also helps in the reduction of food wastage as it first shows the recipes, which contain the ingredients that are expiring soon and matches with users' taste preferences. It also considers that recipes do not violate users' health restrictions and nutritional goals, and use the maximum number of available ingredients in users' kitchen. The proposed system gathers users' taste preferences by exploiting two third-party social media applications (i.e., Facebook and YouTube) and *collaborative-based* filtering algorithm. This thesis also explores various natural language processing techniques such as text analysis and parts of speech tagging to identify the recipes' preferences and to find the most relevant match for each recipe or ingredient having different names.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Hausi A. Müller, my supervisor, whose motivation, enthusiasm, guidance, and immense knowledge were of inestimable value to me. He guided me in the right direction at whatever point he thought I required it during the research. I consider it an honor to work with him.

Dr. Sudhakar Ganti, for being my committee member and mentor. His door was always open whenever I needed any suggestion or feedback for this research.

Roshni and Miguel, who played a significant role in devising and implementing the ideas expressed in this thesis. This thesis would not have been possible without their help.

everybody at Rigi and Pita research groups, for their advice, feedback, and support. In particular, I would like to acknowledge Lorena Castaneda for sharing her thoughts and valuable feedback and also for providing moral support.

my parents, whose help and support was worth more than I can express on paper. I am forever indebted to them for allowing me to pursue my dream. This journey would not have been possible if not for them, and I dedicate this milestone to them.

Chapter 1

Introduction

This chapter introduces the reader to some of the challenges faced by people every day while selecting healthy recipes, considering their dynamically changing taste preferences and the stock of ingredients in their kitchen. We focus on the scope of social media affecting user preferences [GZR+10]. We also discuss the impact of users' taste preferences that influence food wastage which is a primary concern for the public health of this world [AWdHA+15]. One approach to deal with these problems is to develop a recipes recommender systems that can take into account users' health restrictions and goals, their taste preferences and maximize use of ingredients they have in stock while minimizing food wastage.

1.1 Problem Definition and Motivation

> ***"Food is an integral part of our lives, cultures, and well-being, and is of significant interest to public health [AMW15]"***.

In the past decade, the general work habits of people have changed drastically, raising concerns about their well-being. As evident from the current lifestyle and health records of people around the world, we can see that people are suffering from various lifestyle and diet-related illnesses like obesity, diabetes and heart-related issues and these issues prompt 60% of the total deaths [LSH+97, ELL+12, TE17]. According to Ornish et al., healthy dietary choices will forestall or sometimes reverse these health issues [OBB+90].

According to the *World Health Statistics 2014* , more than 1.9 billion adults were overweight, out of which 600 million were obese. More importantly, these figures tend

to increase [Wor18b]. The population, in general, understands that healthy eating is fundamental to their well-being. It is mainly due to the lack of knowledge about what is healthy to eat while considering people's taste [LSH+97, ELL+12]. For instance, a person suffering from obesity might not have adequate knowledge about what recipes to prepare or what dietary guidelines to refer for the selection of appropriate recipes, which are good for her health by taking taste preferences into consideration. Thus, healthy dietary patterns with people's dynamically changing taste preferences are essential for keeping up a salubrious life, which is quite challenging.

In recent years, social media has achieved a great deal of success having millions of users using sites such as Facebook,[1] YouTube,[2] Twitter[3] and Wordpress[4] which play a crucial role in finding the users' preferences. These sites generally depend upon their users to create and contribute contents; to comment on other users' contents with tags, likes, and ratings; to form online relationships; and to join online communities. Social media and personalized recommender systems can mutually benefit from one another [GZR+10].

These sites are expanding day by day, and the size of their content keeps on growing which eventually prompt an overload of information. Such large amount of availability of personal information can be utilized to create a deeper and more personalized experience for users [GZR+10]. Indeed, even big companies such as VEVO,[5] have integrated with many social media sites like Twitter, Spotify,[6] and YouTube to utilize users' personal information to provide more personalized recommendations [Sar18]. Users' personal content on social media can likewise influence the personal dietary habit of the user [AMW15].

In spite of the advantages of healthy eating, adults do not usually engage in good eating habits, particularly, young adults [ELL+12]. Although people regularly attempt to incorporate healthy eating habits after receiving medical advice, social promotion or participating in disease prevention activities; various hindrances or barriers can be encountered, such as constrained knowledge of cooking recipes, and taste preferences [LSH+97, ELL+12]. Lack of time, however, is one of the most reported barriers, which makes it challenging for users to plan and follow a daily healthy diet.

[1] https://www.facebook.com
[2] https://www.youtube.com
[3] https://twitter.com/?lang=en
[4] https://wordpress.com
[5] https://www.vevo.com
[6] https://www.spotify.com/ca-en

In light of this, numerous medical professionals around the world are concentrating on incorporating IT solutions with health management and encouraging users to use online systems in order to improve their lifestyle in a better way [FB10].

Trattner et al. stated that the food recommendation domain had received the least attention as compared to other domains related to leisure and entertainment. They then argued that more attention should be given to the food recommendation domain due to the fact that humans need food for their existence [TE17]. However, choosing food is a challenging task as we have many different options available [SGT10], [SAK+91].

Ueda et al. conducted a questionnaire with 20 men and women in their 20s to 40s to survey the key considerations for menu planning [UTN11]. The result shows their preferences in the following order: (1) food preferences, (2) nutritional balance, and calories, (3) ingredients they have in stock or they can get easily, (4) easy to cook, and (5) mood.

Whenever users go into their kitchen to cook a meal, it is not easy for them to manually search for the recipes considering their taste preferences, health restrictions and goals, and the ingredients they have in stock as there could be numerous blends.

Alongside unhealthy eating, household food wastage is increasingly becoming a public problem, as future ecological conditions will most likely to be unable to maintain the world's limited resources and secure enough food for the human population. One of the significant reasons for food wastage is the users' taste preferences, i.e., disliking of the food [AWdHA+15].

According to the World Food Programme, we are producing enough food to feed everyone in the world, but regardless of that, over 800 million people across the globe go to sleep hungry every night because of food wastage [Wor18a]. As per the statistics, 1 in 9 people on earth is either starving or malnourished. If we take a quarter of the total wasted food in USA, UK, and Europe each year, we can without much of a stretch unravel the aforementioned starving problem [Oli18].

Food wastage is also destructive to the environment. In terms of land, we require a bigger measure of land territory than the land mass of China for the production of food each year that is ultimately never eaten. What's more, even the land for this production is only accessible after deforestation, extinction of animals and birds species, moving human population away, and degrading the quality of the soil. Moreover, we are using 25% of freshwater on the production of this wasted food [Oli18].

According to Williams et al., food packaging plays a vital part in reducing food

waste; and about 20-25% of it could be related to how people manage to cook food before their expiry date, according to package sizes [WWO+12].

Given all the restrictions above, it appears that an IT system for managing the household food and diet plans can potentially reduce food wastage, thereby helping individuals and families to improve their eating habits, and therefore lessening the incidence of diet-related health issues.

Moreover, the upsurge in human needs and desires to organize food items in the refrigerator led to the emergence of large sized refrigerators [Hof14]. Companies like Samsung[7] and LG[8] invented two-door refrigerators in 1990 and two-sided refrigerators in 2000 [Car18]. However, managing the food items in these refrigerators is still a cumbersome task. Also, users have always been in a predicament about what to prepare from the available ingredients, as there can be many combinations and it is not easy for users to explore manually on the web for the various sorts of recipes from these ingredients. To first solve this, in 2010, the concept of smart refrigerators came into the market. However, these refrigerators use only ingredients' expiry date while recommending recipes to users. Even the recently launched smart refrigerators in 2016 by Samsung does not include the users' preferences which are dynamically changing while recommending recipes [SAM18].

To alleviate the aforementioned barriers several internet-based techniques have continuously been proposed for many years. These applications are recommending recipes based on the context of ingredients, the popularity of recipes within that application space and cuisine style. Users have to manually input their contextual information every time to get the recipe recommendations. However, these solutions do not concentrate on numerous compelling context of the users' such as changing taste preferences and health altogether.

Based on the above problems and motivation, we formulated five Research Questions (RQ) as follows:

- *RQ 1 : How can we exploit users' personal context from third-party social media apps such as YouTube and Facebook to extract users' taste preferences for providing relevant recipe recommendations?*

- *RQ 2 : How can we integrate users' dynamically changing taste preferences gathered from various sources such as YouTube, Facebook and Collaborative filtering*

[7] http://www.samsung.com/us
[8] http://www.lg.com/ca_en

to provide better recipe recommendations?

- *RQ 3 : How can we integrate users' dynamically changing taste preferences and health factors to make better recommendations for a healthier life?*
- *RQ 4 : How can we promote the reduction of food wastage while considering users' dynamically changing taste preferences and health restrictions?*
- *RQ 5 : How can we automate the manual web searching by exploiting users' context to improve their experience by refining the browsing of recipes within a short time?*

1.2 Research Methodology

We're drowning in content.
Recommendations are what we need.

David Pierce

The objective of this research is to formulate a smart application as a proof of concept, which is based on context awareness called CAPRecipes. This application aims to solve the above problems mentioned partially by exploiting users' contexts. CAPRecipes utilizes *kitchen, personal and temporal context* to suggest recipes, which are highly relevant to users.

The kitchen context includes the attributes of ingredients (i.e., expiry date and quantity) that are currently available in users' kitchen (i.e., refrigerator and pantry). The personal context includes the users' recipe likes of their social media such as YouTube and Facebook; users' recipe likes on CAPRecipes as well as suggested recipe likes based on similarity with other similar users; health-related information (i.e., allergies, nutrition information) and strict dietary restrictions or moral concerns (i.e., vegan, vegetarian or gluten-free). The temporal context includes users' current meal-time.

By exploiting these contexts, CAPRecipes is able to provide better recipe recommendations to users, thereby fulfilling health goals and restrictions, maximizing the use of ingredients they have in stock and adapting to dynamically changing taste preferences. With the assistance of this system, users no longer need to manually

search on the web for recipe recommendations. We also hope that users' recipes browsing experiences will be enhanced with this system.

To accomplish this research, we use *natural language processing (NLP)* techniques for classifying the recipe likes from the YouTube and Facebook's eclectic list, and for identifying the most relevant match for each recipe or ingredient retrieved from different sources. In addition, we use the *Spoonacular API* as a recipe database to retrieve the recipes based on users' context. To identify the similarity between users in terms of their taste preferences, we use *collaborative filtering* that helps in suggesting those recipes in which users may have shown interest, results in improving recipe recommendations.

With the help of the recommended list, users can read and watch the instructions of recipes and can also like them through the CAPRecipes interface, which in turns improves their future recipe recommendations.

CAPRecipes interface comprises of four parameters in the form of toggle buttons, which provides the users full control over the personalization. For instance, if a user decides that she wants recommendations without the influence of her taste preferences, she can disable the taste personalization parameter and thus CAPRecipes recommend recipes based on the ingredients' availability with their expiry, and her health restrictions and nutritional goals.

1.3 Thesis Outline

Chapter 2 presents the background explaining the notion of context and context-aware applications. It also presents related research work, highlighting aspects that have been considered to recommend recipes to users.

Chapter 3 demonstrates the architectural view of our proposed context-aware personalized recipes recommender system, called CAPRecipes using user interface design and detailed components architecture.

Chapter 4 presents the implementation of RESTful web service APIs, and algorithms used to understand and integrate dynamically changing users' taste preferences of recipes, gathered by exploiting various sources, for providing recipe recommendations according to users' taste preferences.

Chapter 5 presents the implementation of RESTful web service API, and algorithm used to exploit users' personal, kitchen and temporal contexts for providing personalized recipe recommendations.

Chapter 6 presents the evaluation of CAPRecipes where we assess its usability in terms of efficiency, effectiveness, and user satisfaction using two experiments.

Chapter 7 summarizes the thesis and presents the contributions and ideas for future research.

Chapter 2

Background and Related Work

This chapter describes the background topics of this research: factors that affect users' food choices, the notion of context and context-aware applications, brief description of RESTful web service APIs exploited in this research; and presents prior work in the domain of recipe recommender systems.

2.1 Factors Affecting Food Choices

In this section, we illustrate the essential factors considered by an individual while making food-related choices; and portray the steps taken by the experts of Harvard to promote healthy eating.

"We want what we eat to be healthy and tasty [Spe18]".

In today's world, it has been observed that people like to adopt healthy meal plans [Hew18], but they resist healthy eating because they want to continue eating the food they love. People might assume that healthy food is neither tasty nor something they would love to eat [Lac18, Hew18, Cui18]. The lack of knowledge about how to choose healthy recipes while exploiting their taste preferences and time to cook are some of the reasons that are driving people to pick convenient food rather than necessary food. However, we can prepare healthy recipes without imperiling taste [Hew18].

Choosing what to eat is a complex decision that people make every day. According to a qualitative analysis of food choices, Furst et al. and Falk et al. developed a *Food Choice Model* in 1996, which was further explored by Conners et al. in 2001. In

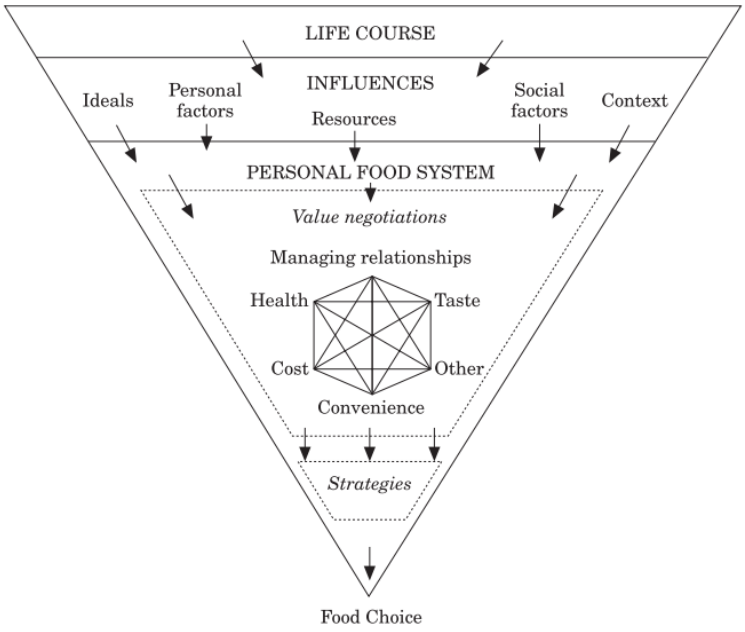


Figure 2.1: Food choice process model [CBSD01]

this model, the events and experiences of individuals are examined over their life course, shaping their food choices, which are influenced by their ideals, personal factors, resources, social relationships, and food context. In view of this model, considerations that individuals weigh when making food-related choices are called *values*. These values are taste, health, cost, time, and social relationships [FCB+96, FBS96, CBSD01]. The Food Choice Process Model presented by Conners et al. is depicted in the Figure 2.1.

As discussed above, people like to adopt healthy eating but planning it is a very complex task. Thus, to tackle the health issues, nutrition experts of Harvard School of Public Health and editors of Harvard Medical School created a *Healthy Eating Plate* outlined in Figure 2.2. This plate provides a simple and trustworthy guide to create a healthy and balanced meal. To promote healthy eating, experts of Harvard have suggested to have a copy of the *Healthy Eating Plate* on refrigerators for people to remember it [oPH18, Pub18].

Users might prefer healthy and tasty food (cf. Section 1.1). As shown in Food Choice Process Model (Figure 2.1), *health*, *taste*, and *social context* are three essential values or factors of a personal food system. In this research, we use these three values since they have a positive impact on the recommendation.

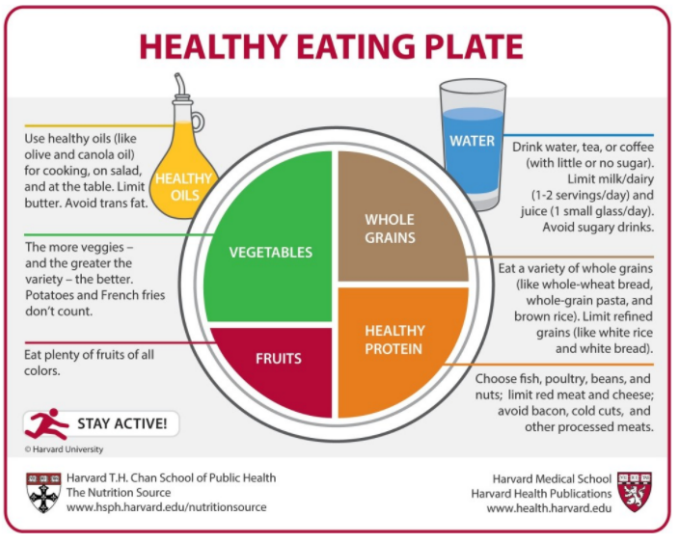


Figure 2.2: Healthy Eating Plate by Harvard [oPH18]

2.2 Context-Awareness

Over the years, researchers made several attempts to define the notion of context awareness. Schilit et al. defined the meaning of context-awareness as a new class of computer software applications that exploit changes in users' environments to change their behavior accordingly [SAW94, MAD00]. Pascoe et al. presented context awareness as the capability of computing devices to detect, sense, interpret and respond according to users' local environment and computing devices themselves [PRM00, BHS04].

According to Dey et al., context-awareness is the utilization of context to provide relevant information and services to users, where the relevance relies upon users' tasks [BHS04, ADB+99, Dey16]. They observed that their definition is more general and does not fit in the establishment of context-aware applications. Therefore, they categorized the features for context-aware applications, included in Section 2.2.2 [Dey16, ADB+99].

2.2.1 What is Context?

Context is becoming progressively more important for ubiquitous computing, where the context of users is constantly changing. Context can be specified as any information in a given situation where an entity exists, or an event occurs; where an entity can be a person, place, or physical or computational object [BHS04, Dey00].

The definition of context has been introduced and defined by many researchers over the years since 1994. Schilit and Theimer were the first researchers who originated and presented a formalization of user-centered understanding of context based on three aspects: where you are, who you are with, and what resources are nearby [SAW94, BHS04]. Also, they stated that context is the environment, which is constantly changing. The following elements of the environment are discussed [Dey16]:

1. **Computing Environment:** connectivity, available processes, accessible devices for user input and display, costs of computing, and network capacity.

2. **User Environment:** set of nearby people, location, and social circumstances.

3. **Physical Environment:** lighting and noise level.

In 1998, Dey et al. and Pascoe et al. redefined the concept of context. The former defined the context as the user's physical, social, emotional, or informational state. The latter defined context to be the subset of physical and conceptual state of interest to a particular entity. Benerecetti et al. classified context into two categories: physical context and cultural context. Physical context incorporates physical features of the environment such as location and condition. Cultural context incorporates a user's personal data, social and belief information [BBB01].

According to Dey et al., context can be defined as the environment, which is relevant to an application and its sets of users. According to them, as situations are dynamically changing it is difficult to find which aspects are important for a particular situation. In addition, they stated that context can include both the implicit and the explicit input. Like in the case of user identification, the identity of a user can be implicitly sensed with the help of facial recognition; or the user can explicitly type his/her name via the keyboard [Dey16].

Bisgaard et al. surveyed the context for investigating common characteristics of how context is defined. Based on their survey, they argued that most of the researchers seem to have an agreement on location, time, identity and environment as key factors summarized in Table 2.1 [BHS04].

After considering all of the aforementioned definitions of contexts, we have concluded that the definition given by Dey et al. is the most relevant to our research. In particular, users' context can be extracted using two types of input: implicit and explicit. In our research, the implicit context includes dynamically changing users' taste

	Location	Time	Identity	Environment	Social Setting	Network	Season	History	Task/Activity	Device
Ryan et al.	✓	✓	✓	✓						
Brown et al.	✓	✓	✓	✓			✓			
Schilit et al.	✓		✓	✓	✓	✓				
Rahlff et al.	✓	✓	✓	✓					✓	
Beale et al.								✓		
Hess et al.	✓	✓	✓	✓					✓	✓
Dey et al.	✓			✓						

Table 2.1: Common characteristics of context [BHS04]

preferences whereas explicit context includes users' health restrictions and nutritional goals; and available ingredients in a users' kitchen.

2.2.2 Categorization of Features of Context-Aware Application

To more concretely represent the field of context-aware applications, Dey et al. presented the categorization of features of a context-aware application. This categorization helps application designers in the development of context-aware applications. Before this categorization, two attempts have been made by Schilit et al. and Pascoe. Schilit et al. defined the categorization into two orthogonal dimensions [Dey16, ADB+99]:

- *Whether the task is to obtain information or to execute a command;*
- *And whether the task is executed manually or automatically.*

These two dimensions were then categorized into four types [Dey16, ADB+99]:

1. **Proximate Selection:** Retrieve the information manually for the user, based on the context availability

2. **Automatic Contextual Reconfiguration:** Retrieve the information automatically for the user, based on their context availability

3. **Contextual Command:** Executes commands manually for the user, based on their context availability

4. **Context-Triggered Actions:** Executes commands automatically for the user, based on their context availability

In the second attempt, Pascoe in 1998 proposed a categorization that was considerably overlapped by the categorization of Schilit et al., but it includes some recognizable differences as well. The former was aimed at identifying the core features of context awareness whereas the latter focused on the identification of context-aware applications. Dey et al. stated that Pascoe's categorization is based on the features of context awareness and maps well with the categorization of Schilit, which is based on the classes of applications. Pascoe's categorization is defined below [Dey16, ADB+99]:

1. **Contextual Sensing:** is the capability to recognize contextual information and exhibiting it to the user. Schilit et al.'s *Proximate Selection* maps well with this categorization.

2. **Contextual Adaptation:** is the capability to execute or modify a service automatically considering the current context, which maps well with Schilit et al.'s *Context-Triggered Actions*.

3. **Contextual Resources Discovery:** is the capability for allowing context-aware applications to locate and exploit resources and services, which are relevant to user's context, and maps with Schilit et al.'s *Automatic Contextual Reconfiguration*.

4. **Contextual Augmentation:** is the capability to incorporate digital data with user's context and did not map with any of the Schilit et al.'s categorization. According to this categorization, a user can view the data whenever she enters in the environment of associated context. For example, a user creates a virtual note providing the details of his/her broken car and attached it to his/her car, which is parked in the garage. The user's family members will then get to know that the car is broken whenever they go into the garage.

Dey et al. discovered that Pascoe's categorization does not reveal one of Schilit's categorization, which is contextual commands.

Later in 2000, Dey and Abowd combined the categorizations of Schilit et al. and Pascoe, and introduced three differences. They defined three categories for their categorization and described them as context-awareness behaviors that an application might exhibit [ADB+99, Dey16]:

- *Presentation of information and services to a user.*
- *Automatic execution of a service.*
- *Tagging of context to information for later retrieval*

In the following section, we describe how context can be modeled for the development of context-aware recommender systems with some popular existing context-aware applications. We also explain standard techniques used for developing recommender systems.

2.3 Context-Aware Recommender Systems

Recommender systems are software applications and tools that provide recommendations to users for several items. These systems have been researched and deployed extensively over the last decade in numerous application areas such as e-commerce, information retrieval, healthcare, ubiquitous and mobile computing, data mining, marketing, or management. [VDLG10, VMO+12]. The recommendation process relates to various decision-making processes, like what items to buy, what music to listen to, or what online news to read [RRS11]. Traditional recommender systems do not consider users' contextual information while recommending the items [AT10].

Researchers and developers perceived contextual information as a relevant factor for personalized recommendations [AT10]. Adomavicius et al. stated that contextual information matters in recommender systems as it has a direct impact on the relevance of recommendations [VDLG10]. Context-aware systems differ from traditional recommender systems because it gathers users' contextual information from their environments and adapts the behavior of the systems accordingly [BDR07]. Moreover, contextual information is system dependent for any particular user, because this information might be considered invaluable for another system [TD09]. Some examples

of context-aware recommender systems are:

CyberGuide was the first attempt to create a context-aware system, which was designed in the year 1996 by Abowd et al.; this system uses location as context and aims to work as a citywide guide, helping tourists visiting places they would like to see [LKAA96, BHS04].

Shopping Assistant exploits users' location and identity to help them in making their personal shopping list at home before going to the grocery store. Whenever a user enters a store, the shopping assistant system provides a map highlighting all the locations of the items present in the user's list. In addition, while the user walks the aisles, the system displays available offers according to the user's context [NPS03, BHS04].

Sourcetone Interactive Radio[1] is a music company which incorporates contextual information in their recommendation engine. To improve the music recommendations to users, they use current users' mood as a context, which users have to stipulate [AT10].

2.3.1 Standard Recommendation Techniques

There are three strategies commonly used by most of the recommender systems: collaborative filtering, content-based filtering, and hybrid-based filtering [BS97, VDLG10].

Content-Based Filtering recommends items based on users' interests, which are usually specified in users' profiles.

Collaborative Filtering identifies the similarity between users and produces new recommendations based on inter-user comparisons.

Hybrid-Based Filtering incorporates both the above techniques.

In our research, we use hybrid-based approach including the natural language processing techniques to provide highly relevant recipe recommendations to users. The

[1] http://www.sourcetone.com

detailed description of how we have integrated these approaches in our recommendation engine to provide recipe recommendations is discussed in Chapters 4 and 5. The following section illustrates RESTful web service APIs that are used for the implementation of CAPRecipes.

2.4 RESTful Web Service APIs

In today's world, most of the projects or applications use REST APIs for the creation of professional services. Companies like Twitter, Google, YouTube and Facebook generate business by utilizing REST APIs. REST [2] architecture is the most logical, efficient and widespread standard in the creation of APIs for Internet services [BBV18].

The term REST was introduced by Roy Fielding in 2000 in his dissertation and follows an architectural style for distributed hypermedia applications [Tut18a, Vaq18]. REST is mainly used to build web services that are lightweight, maintainable and scalable. Services based on the REST architecture are known as RESTful web services [Vaq18]. REST APIs allow developers to perform requests and receive responses via the HTTP protocol such as GET and POST. The principal benefit of using REST is that it can easily connect machines using simple HTTP calls instead of using the complex mechanisms such as CORBA, RPC or SOAP [Web, BBV18] and are language independent and easy to test [Dee18].

Table 2.2 demonstrates various HTTP methods used in RESTful web services [Tut18b].

POST	To create new resource
GET	To read or retrieve a representation of a resource
PUT	To update existing or create a new resource
PATCH	To update or modify existing resource
DELETE	To Delete an existing resource

Table 2.2: HTTP methods of RESTful web services

The RESTful web services often returns JSON or XML as a response type but are not limited to these types only. With the help of HTTP accept header, we can

[2] REST is an acronym for REpresentational State Transfer

easily stipulate a resource type in which we are interested in, and the server returns the resource by specifying content-type of the resource it is serving [Web].

For the development of CAPRecipes, we use various REST APIs that are addressed in the following subsections. We retrieve data in JSON format from all REST APIs used in this research as it has lower-overhead than XML and focuses more on content rather than formatting [Ben18].

2.4.1 YouTube API

YouTube API empowers developers to access users' data and video statistics through a REST API call. It provides feasibility to developers to incorporate YouTube experience into their software applications. YouTube renders various API and tools such as Player API, Analytics API, or Data API [You18a].

According to YouTube, every minute 100 hours of videos are uploaded by users [Arn18]. With the assistant of YouTube Data API (v3), we can request as well as store information regarding these videos, which can be used to provide personalization to users. This API helps developers to fetch search results and to retrieve, insert, update, and delete resources like videos or playlists [You18a, Goo18b].

In this research, we use YouTube Data API to retrieve the list of users' liked videos, which helps in understanding the recipes' taste preferences of users for providing personalized recipe recommendations.

2.4.2 Facebook Graph API

Facebook Graph API is the core of Facebook's platform and enables third-party developers to create their services and applications that will access users' data to read and write from Facebook. This API is extremely advanced and empowers developers to access every possible object that includes users' identity, photos, status, videos, likes, conversations and even their inter-linked relationship with each other [Arn18, Fac18a].

Facebook is the largest social networking site having over one billion active monthly users who spend a quality amount of their time on Facebook. Therefore it contains an invaluable mine of data and personal information of users [Arn18]. This is the principal reason for using this API in this research. Not to mention, for many people, the term Internet is basically Facebook [Pie18].

As Facebook contains such tremendous amount of personal data, we can use this data to create a deeper, personalized experience for users. With the help of Facebook's single sign-on authentication mechanism, developers can access this data on behalf of Facebook users via the Graph API across the web, mobile and desktop apps [Arn18, Fac18a].

Similar to YouTube, we also retrieve users' likes on Facebook in JSON format using Facebook Graph API.

YouTube and Facebook likes are eclectic lists of likes. Therefore to extract recipe likes from this data, we use the natural language text analysis API, explained below.

2.4.3 Natural language Processing API

Natural Language Processing is the realm of artificial intelligence that enables computer softwares to analyze and comprehend human language [Inv18]. It can extract significant insights in the form of structured data after processing, examining and understanding unstructured or semi-structured data gathered from human-generated text and content. The unstructured or semi-structured data could be queries, email communications, social media, videos, customer reviews, or customer support requests [Tec18a].

As discussed in the previous section, we get eclectic list of likes from users' YouTube and Facebook profile. Thus, to extract recipe likes from this list, we incorporate *AYLIEN Text Analysis API*. This API is a package with natural language processing and machine learning-powered tools that can analyze and extract diverse classes of necessary information from text and images [Ayl18a].

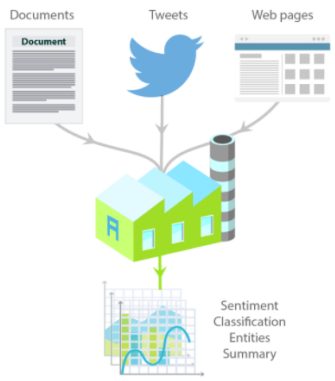


Figure 2.3: High level view of the process of the AYLIEN Text Analysis API

We apply the *classification by taxonomy* technique of AYLIEN Text Analysis API that analyzes contents according to a set of predefined classification procedures and taxonomies. These taxonomies have a tree-like structure, which recursively permit us to traverse from child to parent categories efficiently. There are two taxonomies provided by this API, which are IPTC Subject Codes, and IAB QAG discussed below [Ayl18b].

- *IPTC Newscodes* - International standard for categorizing news content
- *IAB QAG* - The Interactive Advertising Bureau's quality guidelines for classifying ads

2.4.4 Spoonacular Food and Recipe API

Spoonacular API is the Recipe, Nutrition and Food API that has an immense amount of data allowing users to access over 365,000 recipes and 86,000 food products. The food ontology of this API uses natural language processing techniques and lets developers search for recipes according to users' ingredients and health restrictions. This ontology also allows developers to get recipes based on users' special dietary requirements such as vegan, vegetarian or pescetarian [Rap18b, Mas18, Pro18].

This REST API offers food parsing, matching and searching technology for individual app developers as well as large enterprises to create stunning applications with their data. Due to its effective results, it is used by many universities, schools, and hackathons around the world as depicted in Figure 2.4 [Spo18].

Figure 2.4: Countries currently using the Spoonacular API

Overall it is a commercial product however for hackathons and academic purposes an individual can get access to a meager price [Spo18]. Currently, Mashape[3] and Rapid[4] are two companies who merged and became the world's largest API mar-

[3] https://market.mashape.com/dashboard
[4] https://rapidapi.com/

ketplace with 7,500+ APIs and 370,000+ developers [Rap18a], which includes the management of this API on their platform.

Not to mention, we use this REST API to get recipes based on users' available ingredients, and their health restrictions and nutritional goals. The detailed description of all these four APIs and its integration with the CAPRecipes recommendation engine is illustrated in Chapter 4 and 5.

2.5 Prior Work on Recipes Recommender Systems

We all know, food is necessary for human sustenance and every individual look for healthy recipe recommendations in their day to day life. However, it is very challenging for users to make healthier food choices, due to the constant increase in food varieties [GEFT+15] (cf. Chapter 1). In view of this, numerous researchers around the world have been working to build recipes recommender systems for many years. They aim to provide a system, which tailors the recipes for users not only on what users want to eat but also keeping in mind to nourish them more healthily. However, it is still an under-researched domain when compared to other domains [TE17]. In this section, we describe previous researchers' work that has been done for building recipes recommender systems.

Trattner et al. surveyed 25 research contributions ranging from 1986 to 2017, associated with food recommendations shown in Table 2.3. They stated that these research papers are popular, highly cited, recent and relevant in the domain of food recommendations and elected based on their experience. From their survey, they provided a notable summary of past researches on food recommender systems that highlights most pioneering and novel approaches. Besides, they discussed the challenges, lessons learned, and future directions in designing food recommendation systems [TE17].

In Table 2.3 of Trattner et al.'s survey, the first column represents *Author(s) name*, the second column represents *Algorithm(s)*, which defines various algorithmic approaches used in food recommendations domain such as content-based filtering, collaborative filtering or machine learning classifiers. The third column is *Personalized*, which depicts whether author(s) used personalization or not in their recommendation systems. The fourth column, *RecsSys Type(s)*, describes the type of recommendation systems. The fifth column, *Feedback*, shows the means through which users' preferences are taken as feedback to improve recommendations in future. The sixth

Author(s)	Algorithm(s)	Person-alized	RecSys Type(s)	Feedback	Context Feature(s)	Die-tary Constr-aints	Target	Dataset
Elsweiler et al.(2017)	Logistic Random Forrest Naive Bayes	no	Recipes	Ratings Binary	Title Image Ingredients Nutrition Pop & Appr	no	Single User	AllRecipes
Trattner et al.(2017)	LDA WRFM AR SLIM BPR MostPop User-ItemKNN	yes/no	Recipes Meal Plans	Bookmarks Ratings Comments	WHO-FSA health score	no	Single User	AllRecipes
Cheng et al.(2017)	BPR MostPop	yes/no	Recipes	Ratings	City Size	no	Single User	Kochbar
Yang et al.(2017)	Learning to Rank	yes	Recipes	Binary	Image Embeddings	yes	Single User	Yummly
Rokicki et al.(2016)	UserKNN MostPop	yes/no	Recipes	Ratings	Gender	no	Single User	Kochbar
Ge et al.(2015)	MF CB	yes	Recipes	Ratings Tags	Tags	no	Single User	Wellbeing Diet Book
Elsweiler et al.(2015)	SVD-Hybrid	yes	Meal Plans (Set of recipes)	Ratings	Ingredients	yes	Single User	Quizine
Sano et al.(2015)	UserKNN SVD Hybrid NL-PCA	yes	Groceries	Purchases	Food Categories	no	Single User	Grocery store data
Trevisiol et al.(2014)	UserKNN CB	yes	Menus (Set of dishes)	Binary	Text Sentiment	no	Single User	Yelp
Elahi et al.(2014)	MF	yes	Recipes	Ratings Tags	Tags	no	Group User	Wellbeing Diet Book
Harvey et al.(2013)	CB,CF Logistic Reg. SVD-Hybrid	yes	Recipes	Ratings	Ingredients etc.	no	Single User	Quizine
Teng et al.(2012)	SVM	no	Recipes	Ratings	Ingredients Nutrition Cook effort Cook methods	no	Single User	AllRecipes
Kuo et al.(2012)	Graph-based CB	yes	Menus (Set of recipes)	Tags	Ingredients	no	Single User	Food
El-Dosuky et al.(2012)	CB KB	yes	Food items	Query	Tags	no	Single User	USDA
Freyne et al.(2011)	CF	yes	Meal Plans (Set of recipes)	Ratings	-	no	Single User	Wellbeing Diet Book
Ueta et al.(2011)	KB	yes	Recipes	Query	Tags	no	Single User	Cookpad
van Pinxteren et al.(2011)	CB	yes	Recipes	Cooked recipes	Recipe content features	no	Single User	Smulweb
Freyne et al.(2010)	UserKNN CB Hybrid	yes	Recipes	Ratings	Ingredients	no	Single User	Wellbeing Diet Book
Berkovsky et al.(2010)	UserKNN GroupKNN Hybrid	yes	Recipes	Ratings	-	no	Group User	Wellbeing Diet Book
Aberg et al.(2006)	CF	yes	Meal Plans (Set of recipes)	Ratings	-	yes	Single User	Unknown
Khan et al.(2003)	CBR	yes	Meal Plans	Query	Nutrition Content	yes	Single User	Unknown
Mankoff et al.(2002)	CB	yes	Groceries	Purchases	Food Groups	no	Single User	Grocery store data
Lawrence et al.(2001)	AR CF CB	yes	Groceries	Purchases	Product Class	no	Single User	Grocery store data
Hinrichs et al.(1991)	CBR	yes	Meal Plans	Query	Content	yes	Single User Group User	Unknown
Hammond et al.(1986)	CBR	yes	Single New Recipe	Query	-	yes	Single User	Unknown

Table 2.3: Food recommendation researches surveyed by Trattner et al. [TE17]

column, *Context Feature(s)*, explains the type of context taken by authors(s) to make their recommendations better. The seventh column, *Dietary Constraints*, gives nutrition information. The eighth column, *Target*, shows the end user(s) of the system.

And the final column is *Dataset*, which defines the source from where the data were collected for their research [TE17].

From all this research, Trattner et al. observed that content-based, collaborative, and hybrid-based filtering are the three main standard approaches that have been largely employed in the domain of food recommendations. To compare the research, they first structured their sections around these approaches. Then, they compared on the grounds of context used. Next they compared the research whose recommendations were targeted for group of users. Lastly, they compared research encouraging healthier nutritional choices with the aid of food recommendation systems. From their comparison, they ascertained that incorporating health into recommendations has become the main focus of today's research as it helps in improving health problems and nutritional habits of users [TE17].

Trattner et al. illustrated that users might have complex and restrained needs, in such events the performance of standard approaches are very poor. Additionally, recommendations that users are receiving from these approaches can cause a situation where they might not have the availability of ingredients [TE17].

They also stated that approaches used by researchers to date have attempted to predict users' taste preferences by utilizing users' feedback methods such as ratings, tags, and comments; but the performance attained from these approaches are also poor when compared with other recommendation realms. In this aspect, they mentioned that learning users' preferences remains a focus in the food domain, and we are in need of new approaches or methods which might give promising results. Another key finding is the integration of significant context variables because it was observed from various exploratory data analyses that context has a positive influence on food recommendations. Therefore, they stated that there is a need for incorporating new context variables into the recommendation models. Moreover, various researchers proposed different methods to collate nutrition in the recommendation process, but until now all of them are in the preliminary stage. Finally, they suggested that there is a need to work together in achieving standard baseline approaches and importantly, more online studies to understand how new approaches work as an active system used in naturalistic scenarios. According to them, natural language processing techniques might also be efficient in the process of recommendations.

Apart from the researches covered by Trattner et al. in their survey, we also noticed some additional research in the field of food recommendations, which are explained below.

Suksom et al. developed personalized food recommender system utilizing a knowledge-based framework. The functionality of this system is based on integration of two concepts: *1) Person Concept:* It represents an abstract concept of personal profile, involving both users' health status and their food preferences; *2) Food Concept:* It outlines an abstract concept of food items, including food types, cooking methods, food group and nutrition level [SBT+10].

Ueda et al. introduced a method that concentrates on users' preferences, comprising of users' recipe cooking or browsing history for recipe recommendations. In this approach, they break down the recipes into ingredients and classified them as liked or disliked ingredients. Depending upon the classification, they computed the score of each recipe and used this computed score to provide recommendations [UTN11]. Other research done by Takata et al. uses collaborative filtering to determine the value of metadata (another feature) on the basis of users' mood instead of the recipes' ratings. The metadata value relies on five aspects of a user: body, taste, time, money and modify, which are scaled from -5 to 5 [TUMN16].

Besides, the work done by the above researchers, there are numerous websites and mobile applications available in app stores like *Epicurious*,[5] *AllRecipes*,[6] *SuperCook*,[7] and *My Fridge Food*,[8] recommending recipes to users. Even the big companies use these applications as a built-in feature in their smart refrigerator, for example, Samsung uses Epicurious [Bak18].

In summary, the conclusion given by Trattner et al. and surveys of additional research done by us, we concluded that either current research or applications are not considering the integration of users' health restrictions and taste preferences in their recommendation model, or the research. For example, Mouzhi Ge et al. [GEFT+15], were considering the integration but the factors they exploited for taste preferences are tags and ratings, which gave poor results. Due to these poor outcomes, Trattner et al. stated that learning users' preferences remains a focus, and we are in need of new methods or approaches that provide promising results. In light of this, we also incorporate users' social context in CAPRecipes recommendation engine for enhancing users' recipe recommendations according to their taste preferences, as the social context is an essential factor in the process of food choices for an individual (cf. Section 2.1). Also, the recent technology of smart refrigerators given by Samsung is

[5] https://www.epicurious.com
[6] http://allrecipes.com
[7] http://www.supercook.com
[8] http://myfridgefood.com

not consolidating the dynamically changing users' taste preferences, and are providing recommendations only on explicit context such as ingredients they have in stock and users' health issues.

2.6 Summary

This chapter discussed the essential factors to consider by an individual while making food-related choices. We explained the notion of background topics leveraged in this research such as context, context-awareness, and context-aware recommender systems. Then, we described how context could be modeled for the development of context-aware recommender systems with some existing context-aware applications. We also explained the standard techniques used for the development of recommender systems. Moreover, we illustrated the RESTful web service APIs that are exploited in this research. Finally, we presented the prior work on the domain of recipes recommender systems.

Chapter 3

CAPRecipes Infrastructure

This chapter presents the comprehensive infrastructure of CAPRecipes, our context-aware personalized recipes recommender system, its user interface design and components architecture.

3.1 Overview

We seek to alleviate existing users' concerns related to the selection of healthier food choices while considering their dynamically changing taste preferences, which is affected by culinary choices that are continually expanding. Besides, it is a cumbersome task for users to follow dietary guidelines for their health restrictions and nutritional goals and manually map them with their taste preferences, as well as with the recipes which they can retrieve while browsing the web. To address these issues, we propose CAPRecipes, a novel system to provide personalized recipes by exploiting dynamically changing users' contexts.

Figure 3.1 presents the functional overview of CAPRecipes. End users can enable or disable their personalization at any time according to their needs, and based on this personalization, CAPRecipes update the recipe recommendations.

The recommendation engine of CAPRecipes generates a recipes list based on *context analysis* and uses the Spoonacular API as a recipes database (cf. Section 3.3.8). The type of context CAPRecipes utilizes includes:

1. **Users' Personal Context** is a core part of our proposed system, which makes our research unique in comparison to other traditional recipe recommending systems currently available in literature and market (cf. Section 2.5). It is respon-

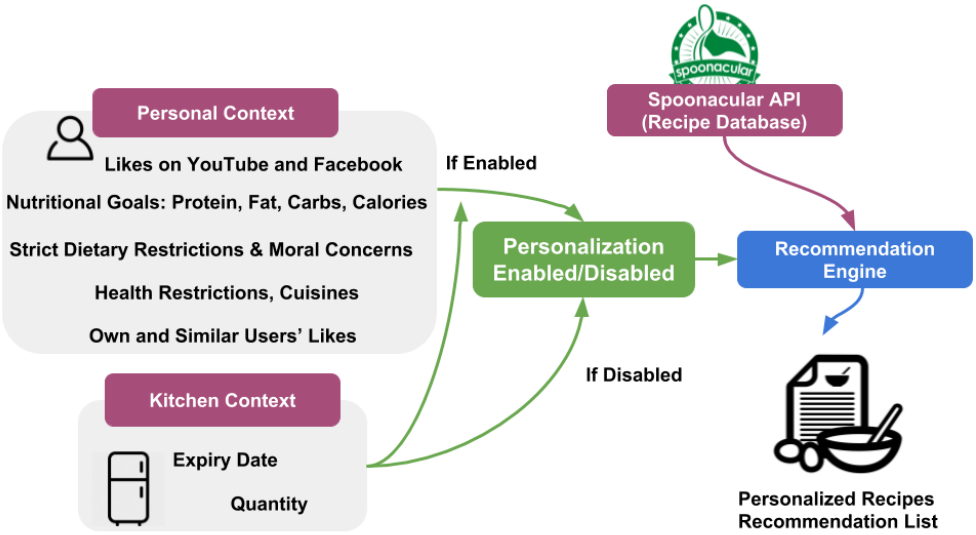


Figure 3.1: Functional overview of CAPRecipes

sible for handling users' dynamically changing preferences. We have classified this context into two types: static and dynamic.

(a) **Static Context** includes users' health restrictions such as allergies; nutritional goals (e.g., eat a certain amount of calories, protein, fat, and carbs); and dietary restrictions or moral concerns like vegan, vegetarian, gluten-free. These are conditions that probably will not change often. Users can enter this context once through the CAPRecipes' web interface and it can be updated anytime.

(b) **Dynamic Context (Taste Preferences)** the purpose of exploiting this context is to make our personalization relevant to users in light of their taste preferences. We gather this context from various sources including:

   i. **Social Media Likes** includes the taste preferences in terms of *likes*, gathered after exploiting users' social media profiles such as *YouTube* and *Facebook*.

   ii. **CAPRecipes Likes** includes the recipe likes that users liked on CAPRecipes.

   iii. **Similar User Likes** includes the suggested likes gathered using *collaborative filtering* based on the similarity measure of a user with other users.

2. **Users' Kitchen Context** includes quantities and expiry dates of ingredients available in the users' kitchen (i.e., refrigerator and pantry). These details are entered by users through the CAPRecipes web interface. The quantities of ingredients will be automatically reduced during meal preparation, according to the recipes' instructions provided in the recommendations.

3. **Users' Temporal Context** includes the users' current meal time detected by CAPRecipes and helps in automatically selecting the dish type, such as main course or breakfast. For instance, assume a user is in the kitchen in the evening and wishes to get a recipe recommendation from CAPRecipes. Hence, CAPRecipes automatically detects her current time and provides a list of dinner recipes. Also, these recommended dinner recipes are according to the user's health restrictions and taste preferences, along with ingredients' availability and their expiry dates.

However, with the help of all these contexts and results which we retrieve from the CAPRecipes, we are confident in stating that CAPRecipes provides highly personalized recipes to users and also enhances users' recipes browsing experience. It eliminates manual steps such as opening a browser, connecting to a website and searching for an appropriate recipe while tailoring them according to their personal contexts. Furthermore, the recommendation process also promotes the reduction of food wastage as recommended recipes represent the users' choice and is also based on the number of soon-to-expire ingredients.

3.2 User Interface (UI) Design

The principal traits and objectives of any software system are exhibited to users via its *user interface*, as it is the only face of software which is evident to users. We explain the user interface design of CAPRecipes in this section to render the notion of our proposed recipes recommender system and all it's intended objectives. In the next section we demonstrate as how to achieve these objectives with the help of a detailed component diagram. CAPRecipes system interaction is attained through a web-based interface which is highly responsive, meaning it is flexible enough to scale on almost any device such as laptop, mobile or tablet.

The user interface design has been developed by keeping in mind the circumstances of users as delineated below:

- users are in the kitchen and want to choose a recipe to cook
- users have a busy schedule
- users have some health restrictions and nutritional goals and want to include them while choosing a recipe
- users would like to maximize the use of ingredients available in their kitchen
- users want to choose a recipe in such a manner that the recipe should uses those ingredients that are expiring soon
- users prefer those recipes that leverage their taste preferences

3.2.1 CAPRecipes Introduction Screen

Figure 3.2 shows the main screen of CAPRecipes through which users can redirect either to the *Login* screen or *Registration* screen.

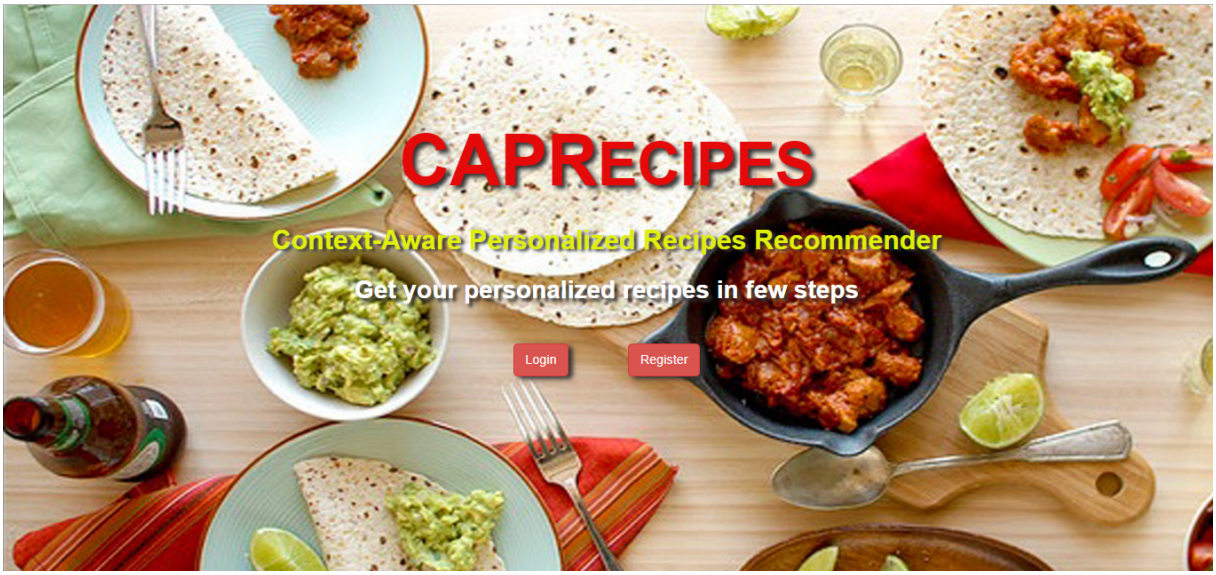


Figure 3.2: CAPRecipes Introduction screen

3.2.2 Registration Screen

If users want to get personalized recipe recommendations, they first need to create an account with the assistance of the screen depicted in Figure 3.3. To attain this, users need to enter a username, email address, password and re-enter the password

for confirmation; and then click the *Register* button, which creates a user account and redirects them to the *Personal Context Gathering* screen.

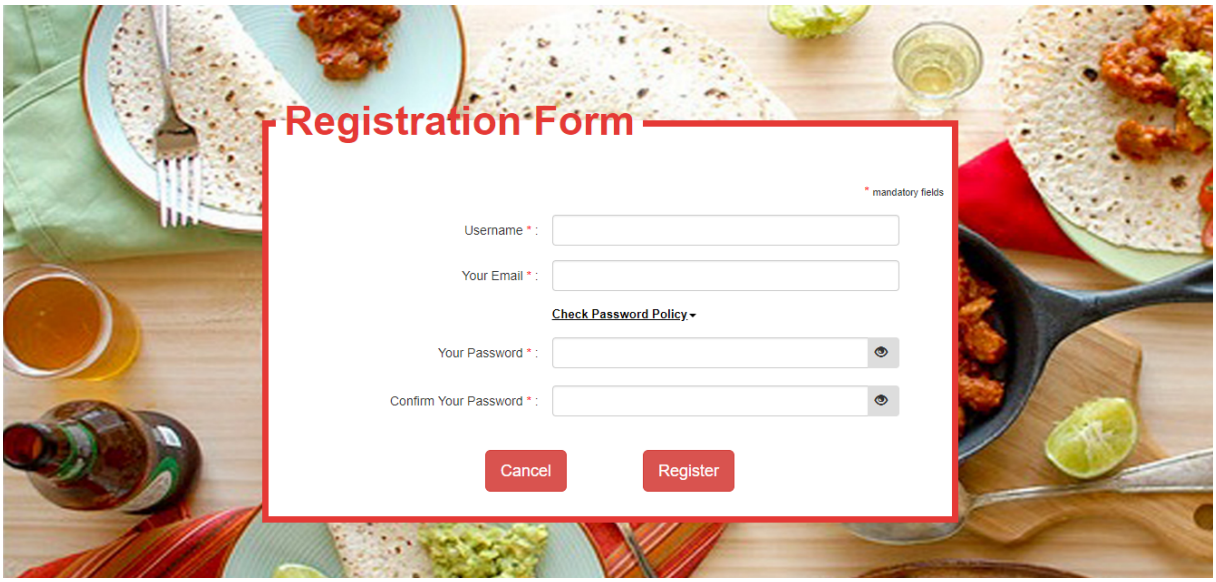


Figure 3.3: CAPRecipes Registration screen

3.2.3 Login Screen

Figure 3.4 shows the screen, which permits users to login into CAPRecipes through their credentials. This page also handles the forgotten password functionality. Once users click the *Login* button, they will be redirected to their *Personalized Recipe Recommendations* screen.

3.2.4 Personal Context Gathering Screen

Figure 3.5 is the principal screen in our recommendation process and it collects all the users' contexts needed for providing personalized recipe recommendations. Users are redirected to this screen in two ways:

1. **From Registration Screen:** when users create their account for the first time,
2. **From My Profile Tab:** when users are already logged in and wish to change their context, which influences their personalized recipe recommendations accordingly. *My Profile* tab is shown in Figure 3.6.

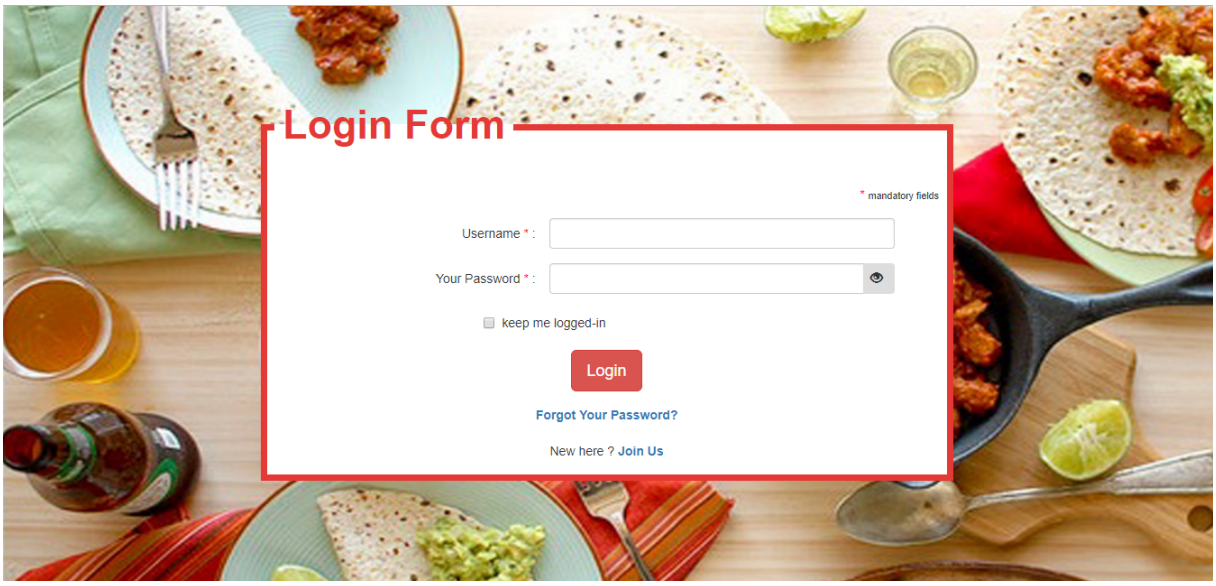


Figure 3.4: CAPRecipes Login screen

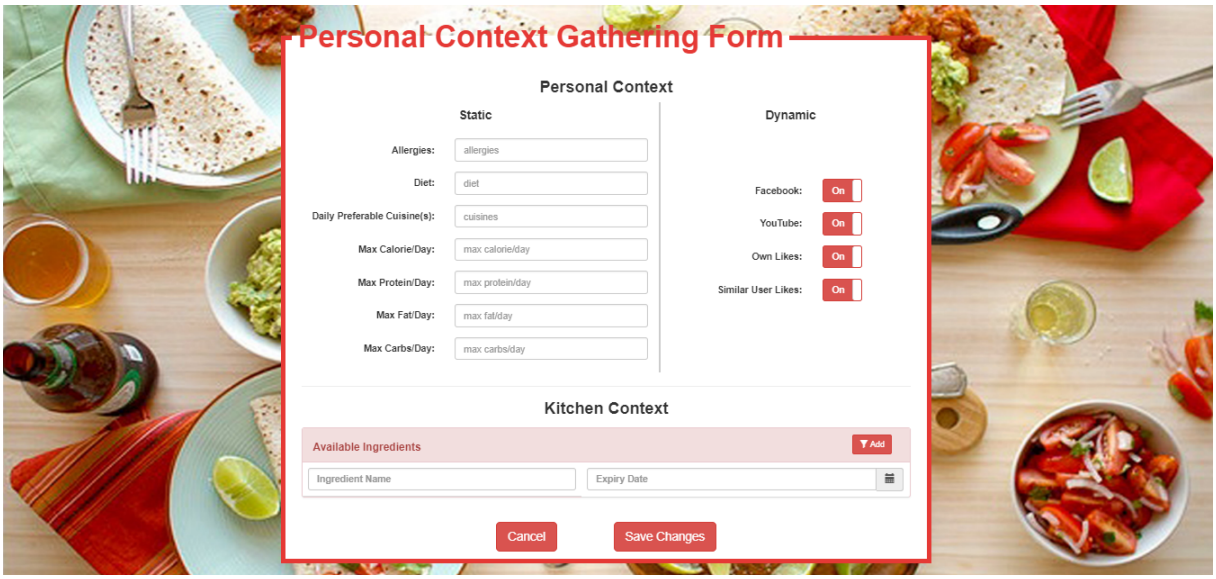


Figure 3.5: CAPRecipes Personal Context Gathering screen

The users' contexts for CAPRecipes are partitioned into two categories: *Personal* and *Kitchen*. Personal is further categorized as: *Static* and *Dynamic* (cf. Section 3.1).

Context Gathering and Generation

1. **Users' Personal Static Context:** As shown in Figure 3.5, users are requested to enter their personal static context. This is the context which does not change

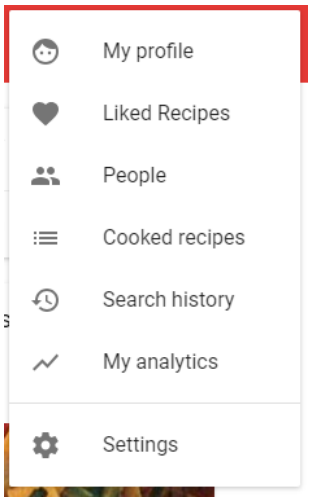


Figure 3.6: CAPRecipes My Profile tab

frequently and thus only updated when users' health condition or moral concerns change. It consists of:

- **Allergies:** Users can enter the list of ingredients, in a comma-separated form, to which they are allergic such as egg, shellfish, soy, or wheat.
- **Diet:** Users can enter the diet type to which the recipes must be compliant to, such as pescetarian, vegan, or vegetarian.
- **Cuisine(s):** Users can choose one or more cuisine types, which they usually want in their recommendations such as African, Chinese, Japanese, Indian, American, or German.
- **Nutritional Information:** Users can enter their nutritional information according to their goals. The nutritional information includes *max calories/day*, *max protein/day*, *max fat/day* and *max carbs/day*.

2. **Users' Personal Dynamic Context:** As portrayed in Figure 3.5, we have four toggle buttons by which CAPRecipes gathers users' dynamic taste preferences from various sources. These toggle buttons are:

   - **Facebook:** Once users turn ON this button, CAPRecipes redirects to the Facebook permission/authorization page, shown in Figure 3.7. After users permit the access, CAPRecipes obtains the users' personal likes from their Facebook profile.

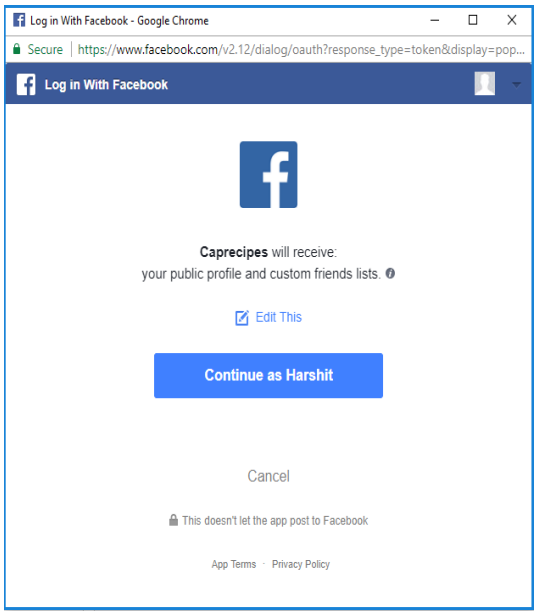


Figure 3.7: Facebook authorization page for CAPRecipes

- **YouTube:** Similar to Facebook, once users turn ON this button, users are redirected to YouTube permission/authorization page, shown in Figure 3.8. After allowing the access, CAPRecipes obtains the users' personal likes from their YouTube profile.

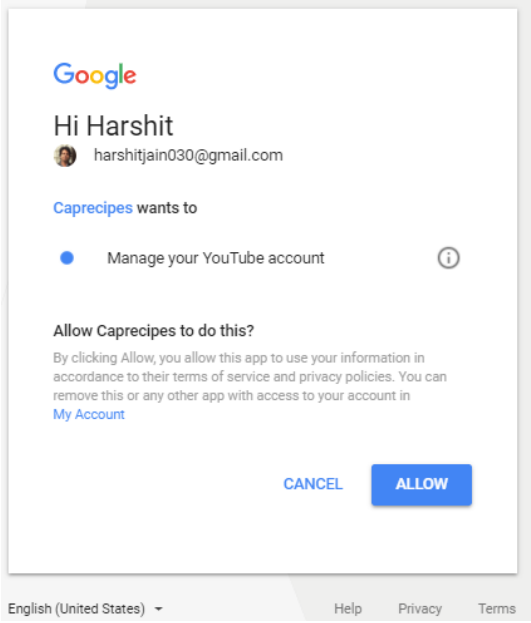


Figure 3.8: YouTube authorization page for CAPRecipes

- **Own Likes on CAPRecipes:** By turning ON this button, recipes liked by users on CAPRecipes interface are considered for the recommendations.
- **Similar User likes on CAPRecipes:** By turning ON this button, CAPRecipes uses suggested likes for recipes retrieved from similar users.

3. **Users' Kitchen Context:** Figure 3.5 exhibits the *Personal Context Gathering* screen where users are requested to enter ingredients with their expiry dates and quantities available in their kitchen. Though users do not need to update these details every time because the quantity of ingredients that utilizes as a part of preparation will be automatically reduced according to recipe's instructions provided by CAPRecipes on the *Recipe Instruction* screen (cf. Figure 3.12). Manually entering these details is a tedious task for users in their hectic schedule. Indeed, even with the recent launch of the smart refrigerator by Samsung, users need to enter these ingredients' details manually [Gri18, Cri18b]. Therefore, incorporating an automated gathering for these details is still a significant challenge. To overcome this, we have encountered two approaches as follows:

   - **Grocery Receipt Recognition:** With the help of smart camera embedded in users' IoT devices such as mobile and tablet, users can scan their physical receipts after purchasing groceries, and thus using advanced algorithms, we can analyze the receipts, and ingredients will be extracted and added automatically with their general expiry dates in users' kitchen context. The general expiry dates of the ingredients can be retrieved from various suggested sources such as Government of Canada [oC18], University of Nebraska-Lincoln [Foo18].
   - **Online Grocery Shopping through CAPRecipes:** In the future, we can add grocery shopping functionally in CAPRecipes, empowering users to purchase groceries from the comfort of their home. After shopping through CAPRecipes, we can extract the ingredients and can add them to a users' kitchen context with general expiry date, which can be retrieved from various sources featured above.

After entering the above contexts, users can click on the save button, and then CAPRecipes redirects them to the *Personalized Recipe Recommendations* screen.

3.2.5 Personalized Recipe Recommendations Screen

Figure 3.9 exhibits the list of personalized recommended recipes gathered by leveraging users' contexts. Users have four options to change their recommendations by allowing all or some of the parameters through toggle buttons, directly available on this screen. These are *Ingredients*, *Health*, *Taste*, and *Ingredients Expiry*. However, it is not possible for users to turn OFF all the parameters, and at least one of them should be turned ON to get the recommendations.

For instance, if a user has turned ON all the parameters other than *health*, then the recommended recipes list will automatically orchestrate according to the number of those available ingredients which are expiring soon from high to low, leveraging user's taste preferences and maximizing the use of ingredients.

This recommendation enhances the users' recipes browsing experience because users

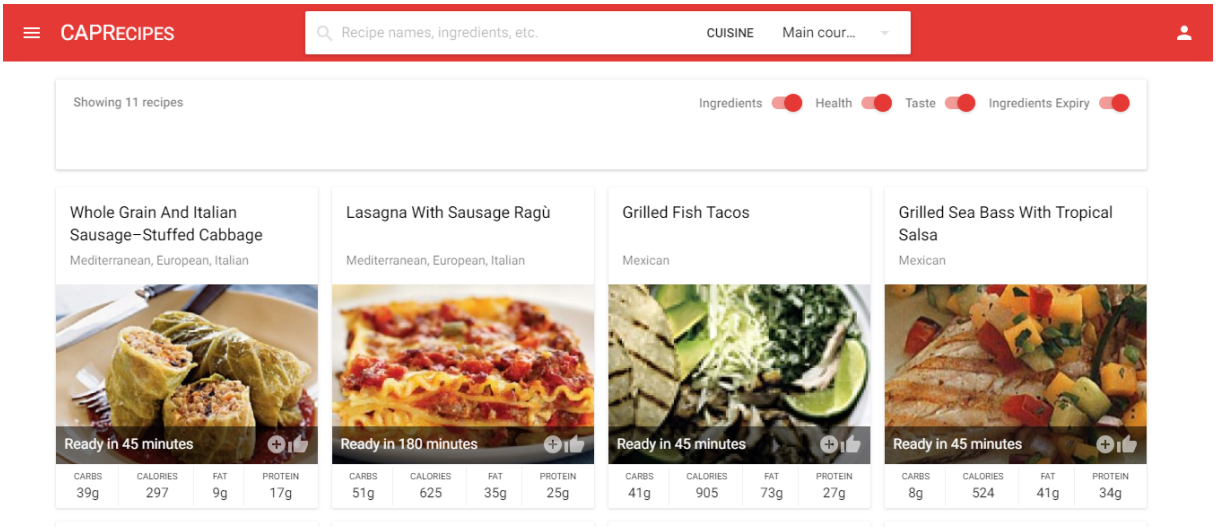


Figure 3.9: CAPRecipes Personalized Recipe Recommendations screen

get their personalized recommendations with only a few steps as well as it considers most of the significant factors, which have been identified as imperative by the research done till now in the field of recipe recommendations (cf. Section 2.5). The information shown for each recipe includes:

- Name
- Image of the recipe
- Cuisine

- Ready time in minutes
- Like button
- Nutrition info. (e.g., Calories, Carbs, Protein, Fat)

Once users choose a recipe which they prefer to cook from the list of recommendations, they can click on it, which redirects them to the *Recipe Instructions* screen; this screen renders the step-by-step instructions/guidelines for the recipe.

The key features provided by the *Personalized Recipe Recommendations* screen are as follows:

- **Dish Selection:** The dish, such as main course, or breakfast, is automatically selected in view of the temporal context of users. Aside from this, if users like to pick any other dish type, they can easily select it using the drop-down menu, depicted in Figure 3.10. For instance, it's 7 in the morning, CAPRecipes analyses the temporal context and recommends breakfast recipes. But in this case, the user needs to go to the office with her lunch. For this, she can manually select lunch recipes using the drop-down menu.

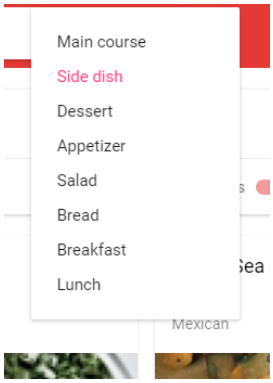


Figure 3.10: CAPRecipes Dish Selection Menu screen

- **Cuisine(s) Selection:** For the personalized recipe recommendations, CAPRecipes automatically sets the cuisine(s), which users have entered on the *Personal Context Gathering* screen. Regardless of that, users have the flexibility to select or alter any different cuisines from the list at runtime, as shown in Figure 3.11, modifying their recommendations accordingly. Having said that, these alteration of cuisines are only for that particular session, meaning it does

not update users' context permanently. For instance, a user retrieves the recommendations of Mexican and Japanese recipes, as CAPRecipes extracted these two cuisines from a user's personal context, due to the fact that this is what user usually prefers to eat. But if the user desires to eat Chinese cuisine, user can select Chinese cuisine from the list and can unselect the previously selected cuisines. This action will render Chinese recipe recommendations based on ingredients available while incorporating user's health restrictions and taste preferences for Chinese food.

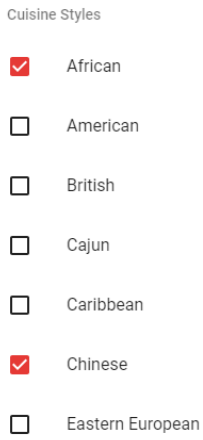


Figure 3.11: CAPRecipes Cuisine(s) Selection Menu screen

- **Explicitly Searching for Recipes:** CAPRecipes offers an additional feature that lets users to explicitly search for any recipe which they like to cook. Users are redirected to the *Recipe Instructions* screen of the searched recipe, which displays all the information and functionalities provided by the *Recipe Instructions* screen.

3.2.6 Recipe Instructions Screen

Figure 3.12 provides the set of instructions for cooking a recipe. These instructions comprise ingredients and types of equipment used in each step for preparing a recipe. Additionally, this screen also exhibits the detailed information for each recipe such as *title*, *cuisine*, *preparation time*, and *nutritional information*.

The key features provided by the *Recipe Instructions* screen are as follows:

- **I Like It:** Users click this button if they like a recipe, which serves as a

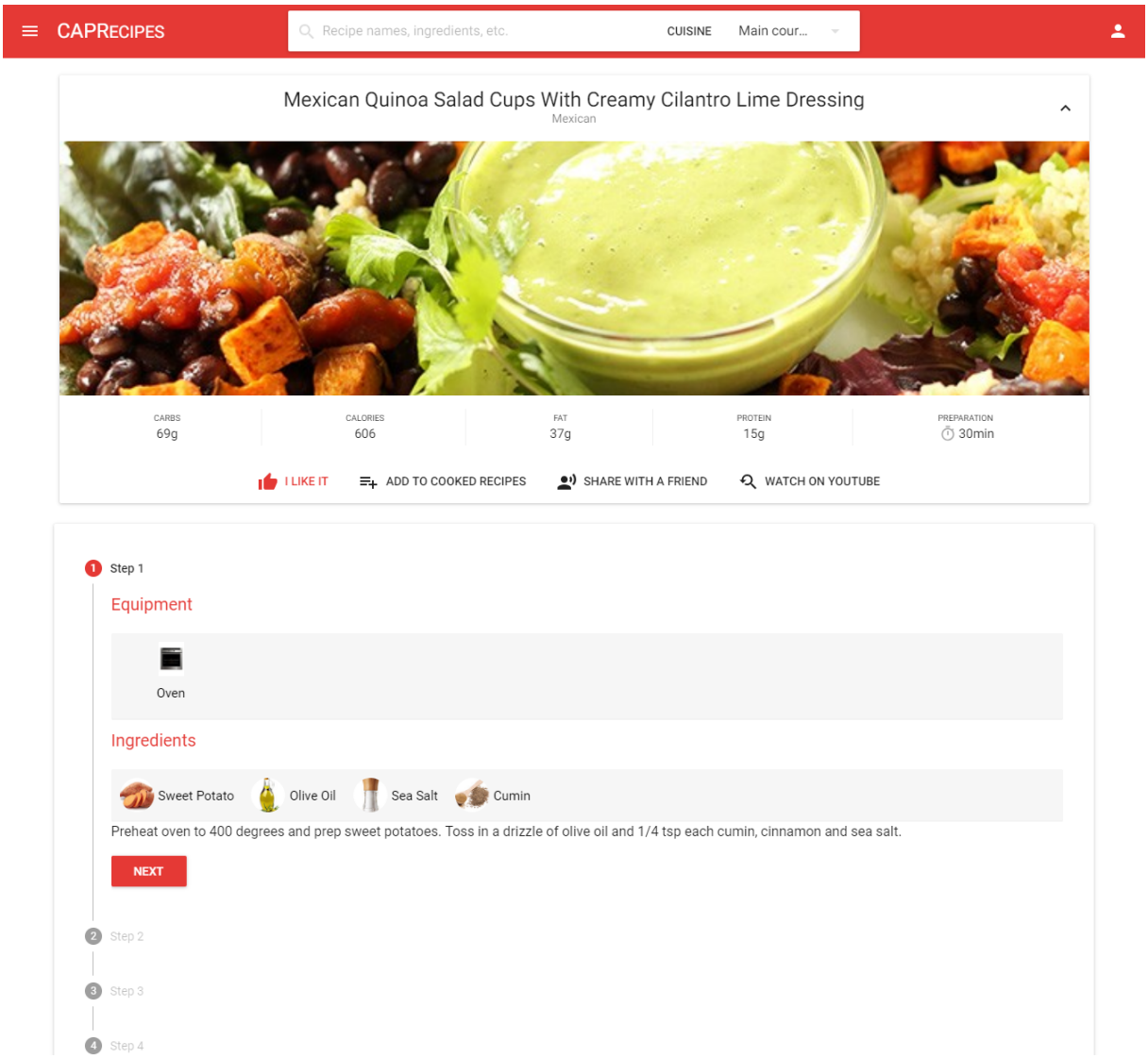


Figure 3.12: CAPRecipes Recipe Instructions screen

feedback for CAPRecipes recommendation engine to enhance future recipe recommendations.

- **Add to Cooked Recipes:** Users click this button to add a recipe to their prepared recipes list. This list displays all the recipes users have prepared.
- **Share with a Friend:** This feature empowers users to share a recipe with their friends on social platforms such as Facebook, which is helpful in enhancing the social relationship of users.
- **Watch on YouTube:** If users prefer to watch the instructions instead of

reading it from the *Recipe Instructions* screen, it can be achieved by clicking this button, which redirects users on YouTube to show an instructional video for preparing a recipe.

3.3 Software Architecture and Components

This section encapsulates the comprehensive overview of CAPRecipes architecture, which comprises nine software components. In these nine components, six have been categorized as core components, and three are external web service APIs depicted in Figure 3.13. The six core components are: CAPRecipes GUI, Server Component, Users' Context Component, Matching Component, Similar User Component, and CAPRecipes Recommender Component. The three external service components are Recipe Component, Social Media Component, and Text Analysis Component. The implementation and integration of these components are discussed in Chapter 4 and 5.

3.3.1 CAPRecipes GUI

GUI stands for a Graphical User Interface allowing users to render input via tangible devices such as mobile, tablet, keyboard, or mouse, and displays output on the screen [Use18]. CAPRecipes GUI is a client program that follows a client/server model. CAPRecipes provides several functionalities such as login, registration, enabling and disabling personal preferences, sharing recipes with friends; and displays the personalized recommended recipes list. Section 3.2 outlined the description of the user interface design of CAPRecipes and its functionalities. For requests made by the client, the server calls various other components to achieve highly personalized recipes for users considering their contexts. The principal function of this component is to gather *kitchen* and *personal* context from users, which are then forwarded to the *Users' Context Component* via a server for storing it in a JSON format structure.

3.3.2 Server Component

The Server handles traditional client-server interactions and connects with multiple other components. The Server accepts requests made by clients and then makes a call to other components for rendering relevant recipes on the basis of the users' contexts. The Server is responsible for several fundamental operations such as:

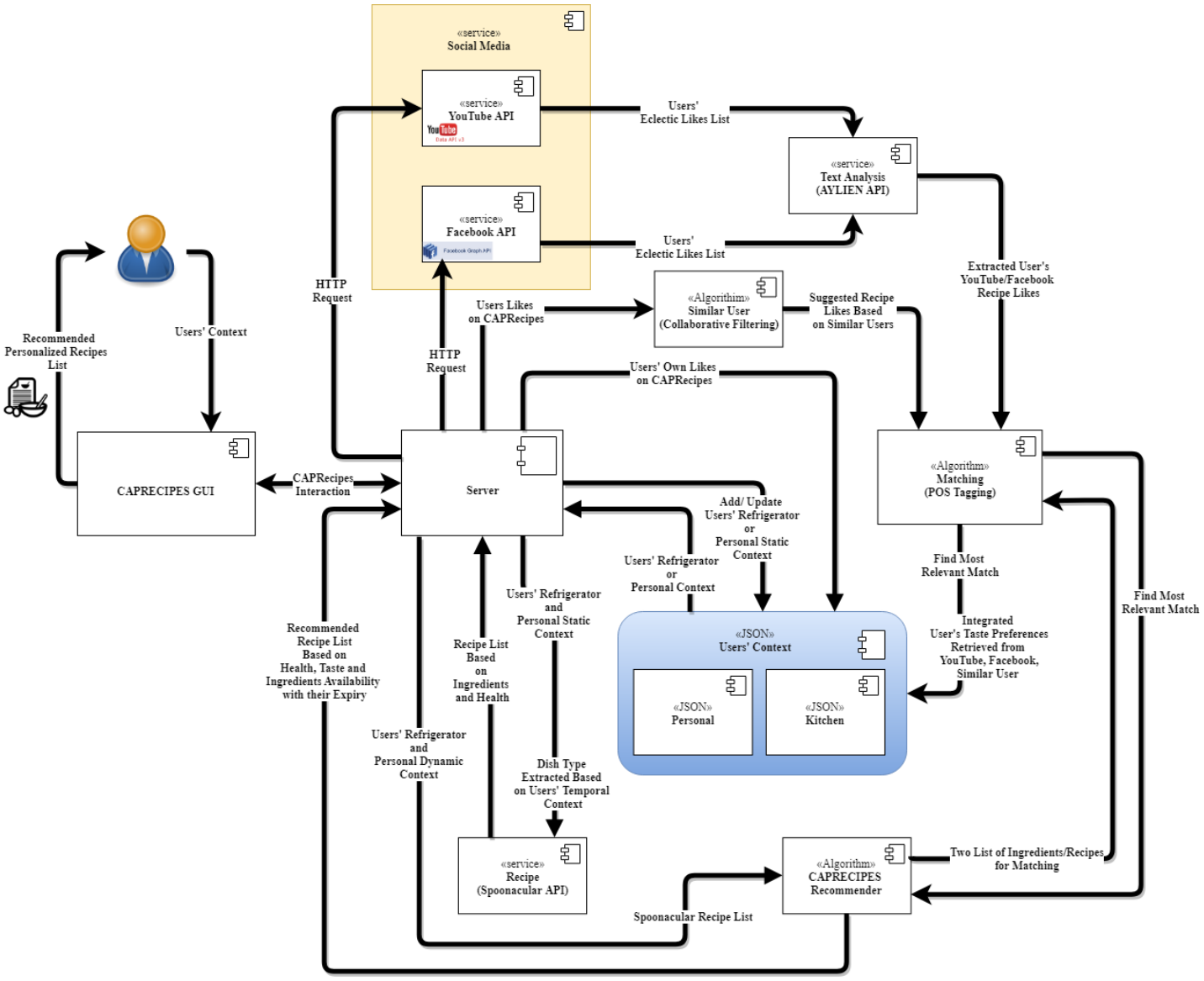


Figure 3.13: Architecture and component diagram of CAPRecipes

- Dynamically updating users' personal dynamic context (i.e., taste preferences) with the assistance of various other components such as *Social Media*, *Similar User*, *Text Analysis*, and *Matching Component*.

- Selecting the dish type automatically (such as main course or lunch), based on the users' current meal time (i.e., temporal context), which will then forwarded to the *Recipe Component* for retrieving a specific type of recipes list.

- Passing users' kitchen and personal static context to the *Recipe Component* for

retrieving the recipes list.

- Passing users' kitchen, personal dynamic context and recipes list retrieved through the *Recipe Component* to the CAPRecipes *Recommender Component*, which provides the final personalized recipes list.

3.3.3 Social Media Component

As argued in Chapter 1, millions of users are using social media applications like *Facebook*, *YouTube*, and *Twitter* and the amount of personal content on these sites is expanding day-by-day which plays a crucial role in finding users' preferences, and can influence personal dietary habits of users [AMW15]. Likewise, this information can be utilized to create a deeper, more personalized experience for users [GZR+10]. Users' preferences can also influence food wastage [AWdHA+15]. Therefore, our research also exploits two of the most popular third-party social media applications for retrieving users' taste preferences with the help of this component. Currently, this component is exploiting Facebook and YouTube, but in the future it is capable of incorporating other social media applications. To get users' taste preferences, the Server calls this component with two separate HTTP calls for any specific user, one for Facebook and another for YouTube. These requests provide results in JSON format after users authorize requests to CAPRecipes. The JSON response contains users' preferences in terms of likes. However, these likes are the eclectic list of likes, therefore, to extract recipe likes from this eclectic list of likes, this component forwards the JSON response to the *Text Analysis Component*.

3.3.4 Text Analysis Component

The Text Analysis Component uses the AYLIEN Text Analysis API service, which analyzes the information for each like and determines if it belongs to a recipe category. For the analysis, two separate HTTP calls, for Facebook and YouTube, are made by this component and return the results in JSON format which are then forwarded to the *Matching Component* for further processing.

3.3.5 Similar User Component

This component analyzes and computes the similarity matrix of a given user with other registered users of CAPRecipes using the *collaborative filtering* algorithm.

This matrix suggests those recipes, which are not in the personal dynamic context of a given user, thereby adding the recipes to a user's personal dynamic context, which also assists in providing personalized recommendations to users according to their taste preferences. The *Collaborative filtering* algorithm generates the result in JSON format for any specific user, which is forwarded to the *Matching Component* for further processing.

3.3.6 Matching Component

According to Trattner et al. and based on our analysis, there are possibilities that any recipe/ingredient can have multiple names, and this is one of the major challenges faced during recipe recommendations [TE17]. Moreover, Emtimov et al. stated that the ingredients having same name on different sources or recipes do not follow the same naming structure because people use human natural language and can have different ways of expressing ingredient names. However, Parts Of Speech tagging (POS tagging) is one of best techniques that can be utilized to find the most relevant match for each recipe or ingredient [ES15]. Therefore, our research also exploits POS tagging technique using this component. It comprises of two subcomponents according to the task, categories are depicted in Figure 3.14.

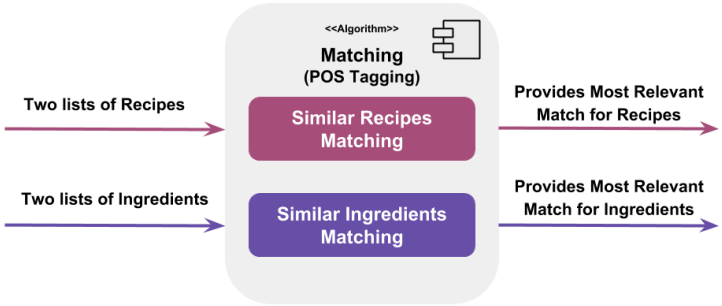


Figure 3.14: Detailed architecture of CAPRecipes Matching component

This component is called by various other components such as *Text Analysis*, *Similar User*, and CAPRecipes *Recommender Component* for finding the most relevant match for each recipe or ingredient, which helps in eliminating duplicate recipes/ingredients or in recommending the recipes based on users' taste preferences.

3.3.7 Users' Context Component

This is the core component of CAPRecipes and is divided into two sub-components: *Kitchen* and *Personal*, which has JSON format. For an illustration, Listing 1 and 2 shows Harshit's kitchen and personal context. The former contains the ingredients details available in users' kitchen, and the latter contains users' personal static context such as health restrictions and nutritional goals, and users' personal dynamic context such as taste preferences. This component is called for various events such as:

```
{
    "Kitchen Context": [
        {
            "Ingredients": {
                "fish": {
                    "id": 1,
                    "Amount in lb": 2.3,
                    "Expiry": "03/30/2018"
                },
                "tortilla chips": {
                    "Expiry": "03/29/2018",
                    "...."
                },
                "salsa": {"...."},
                "pecorino": {"...."},
                "garlic": {"...."},
                "tomatoes": {"...."},
                "cilantro": {"...."},
                "onions": {"...."},
                "lime juice": {"...."},
                "tuna steaks": {"...."},
                "lasagna noodles": {"...."},
                "capsicum": {"...."},
                "cucumber": {"...."}
            }
        }]
}
```

Listing 1: Simplified version of Harshit's kitchen context

- **Retrieving suggested recipes:** users' personal dynamic context is sent to the *Similar User Component*
- **Eliminating the duplicity:** users' personal dynamic context is forwarded to the *Matching Component* with the recipes of users' taste preferences retrieved either from *YouTube*, *Facebook* or *Similar Users*

```
{
    "Static Context": [
        {
            "Allergies": "peanut,shellfish",
            "Cuisine": "mexican,italian,french",
            "Health Goals": {
                "Protein": 98,
                "Carbs": 230,
                "Fat": 74,
                "Calories": 1500
            },
            "Diet": "non-vegetarian"
        }
    ],
    "Dynamic Context": [
        {
            "Recipe Preferences": [
                "Tortilla-Crusted Tilapia with Mango Salsa",
                "Arugula Lasagna",
                "Quinoa Salad",
                "Grilled Tuna with salsa",
                "Nachos",
                "Grilled Tuna With Chipotle Ponzu And Avacado Salsa",
                "Lasagna With Sausage Ragu",
                "Quinoa Enchilada Casserole",
                "Bean Burritos",
                "Grilled Sea Bass with Tropical Salsa",
                "Cremini Mushroom Pasta With Wilted Arugula And Goat
                ↪  Cheese",
                "......"
            ]
        }]
}
```

Listing 2: Simplified version of Harshit's personal context

- **Extracting the list of recipes:** users' kitchen and personal static context are forwarded to the *Recipe Component*
- **Recommending the final personalized recipes:** users' kitchen and personal dynamic context are forwarded to the CAPRecipes *Recommender Component*, which is ultimately the result of this research

3.3.8 Recipe Component

As argued in Chapter 1, individuals are experiencing various well-being related issues such as obesity, diabetes, and heart-related problems and these issues can be prevented and sometimes can be reversed if individuals start eating healthy food. But individuals are unaware of what recipes they can eat given their specific health restrictions. Also, Trattner et al. stated that poor availability of ingredients is a key challenge in food recommendations [TE17]. Therefore, our research tackles the health-related issues and poor availability of ingredients by consolidating the Spoonacular API in the recommendation engine of CAPRecipes with the assistance of this component.

The Spoonacular API is extensively utilized by various researchers and developers around the world because of its accurate results as well as it returns a more diverse collection of recipes compared to the other APIs like Yummly API [JL18]. Therefore, we also use the Spoonacular API instead of creating our recipes database for CAPRecipes.

Every time users want to get recipe recommendations from CAPRecipes, the Server makes an HTTP call to this component with the following information:

users' kitchen context

users' personal static context and

dish type, which is extracted by the Server after exploiting users' temporal context

This component acts as a *content-based* filtering while internally using the natural language processing that filters recipes by giving above contexts and returns the list of recipes with its necessary information in JSON format to the Server, which is then forwarded to the CAPRecipes *Recommender Component* for further processing. The list obtained from this component is based on the users' health restrictions and goals while maximizing the use of ingredients they have in stock.

3.3.9 CAPRecipes Recommender Component

As clarified in Chapter 1 and 2, individuals like to adopt healthy diet plan, but they usually do not engage in it because they assume healthy food is not tasty, and as a result they do not want to stop eating the food they love. The lack of knowledge and

time are some of the main reasons that force people to pick convenient foods rather than healthy foods. Moreover, users' taste preferences (i.e., disliking of food) and not cooking the food before ingredients expiry date triggers the inevitable problem of food wastage, which is one of the leading concerns for the public health of this world. Reducing food wastage is one of our primary goals that would help in securing enough food for future population. Our research solves these problems with the help of this component that uses the *Personalized Recipes Engine* algorithm.

Once we retrieve the list of recipes from the *Recipe Component*, the Server calls this component with three JSON lists, which are orchestrated by leveraging various other components of CAPRecipes. These lists are: recipes list obtained from the *Recipe Component*, users' kitchen, and users' personal dynamic context. Based on these lists, this component further refines and sorts the recipes list of the *Recipe Component* for providing the highly personalized recipes to users by exploiting their taste preferences and expiry of ingredients, which is displayed to users as the CAPRecipes personalized recommendations.

By integrating users' personal, kitchen and temporal context, CAPRecipes deliver more personalized recipe recommendations as compared to other recipe recommender systems while at the same time focuses on minimizing the problem of food wastage.

3.4 CAPRecipes Deployment

To achieve the intended goals of CAPRecipes, we use traditional client-server architecture for implementing the prototype application. We built CAPRecipes in Javascript and deploy using Node.js on our local machine. Node.js is a cross-platform runtime environment for a server-side web application [X+16]. Node.js is lightweight and has powerful JavaScript libraries in building event-based, data-driven, I/O-heavy applications for the Web [Chr18b]. In our development, we perform the following steps to do the deployment:

- To run the CAPRecipes project at localhost:3020, we execute *npm start* from the root directory of our project. Node checks for a *scripts* object in the *package.json* file. If the *package.json* does not have any scripts object, then Node will execute the command node *server.js* instead.

- To build and compile the project, we execute *npm run build* that compiles all

the necessary files to the build folder.

- After successful deployment as illustrated in Figure 3.15, the CAPRecipes Introduction screen is shown as depicted in Figure 3.2.

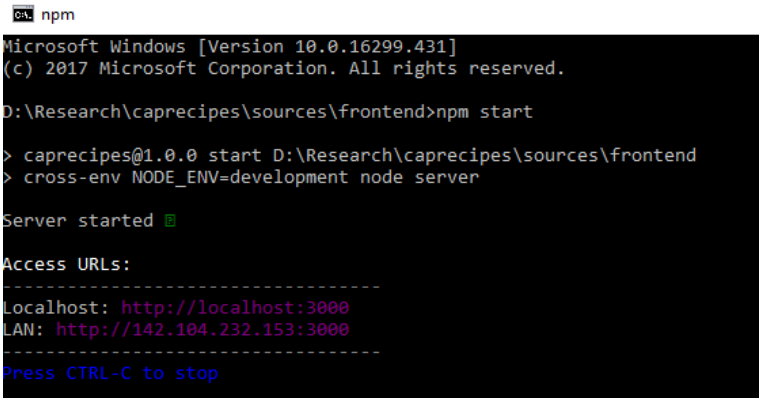


Figure 3.15: CAPRecipes Introduction screen

Over a decade, the use of smartphones has increased and held a significant place in people's lives [Per18]. As a result, people are more inclined towards using mobile applications instead of web-based applications [RWM14]. Thus, our future goal is to make CAPRecipes available to the general public by deploying it as a mobile-based application through the use of cloud computing architecture. So, the people can easily use CAPRecipes app from anywhere, at any time via the internet. Adapting cloud architecture assists in improving the performance and handles the dynamic workloads without failure by scaling the resources on demand.

3.5 Summary

This chapter presented CAPRecipes and its overall architecture. We explained contexts utilized in CAPRecipes for providing the relevant recipes recommendations considering the issues of users, related to the selection of healthier food choices keeping in mind their dynamically changing taste preferences and poor availability of ingredients. Then we described the user interface design with its key features in order to render the notion of CAPRecipes and its intended objectives. Finally, we conceptualized the software architecture of CAPRecipes with its detailed components.

Chapter 4

Understanding and Integrating Users' Dynamic Context

This chapter describes the services and algorithms used to understand and integrate users' recipes taste preferences that change dynamically. These preferences are gathered by exploiting various sources which help in recommending those recipes which matches with users' taste preferences.

4.1 Integrating Social Media Services

In this section, we present the services provided by third-party social media applications for gathering users' taste preferences from their profiles. Moreover, these are the services that are utilized by the *Social Media Component* of CAPRecipes.

4.1.1 YouTube API

Google provides various services, empowering developers to integrate them into their applications. The YouTube Data API is one of these services. To access this service, we first enable it from the *Google Cloud Console* and then create an API key. To access users' data, we request users' consent for which we create an OAuth client ID from the same console. After creating API key and OAuth client ID, developers can download the credentials from the Console in the form of a JSON file named *client_secrets.json* that stores client_id, client_secret, and other OAuth 2.0 parameters [Goo18a]. Listing 3 outlines the simplified version of client_secret.json used for

CAPRecipes. Due to security reasons, *Client Id* and *Client Secret* are not included.

```
{
  "client_id": "CLIENT_ID",
  "project_id": "cap-recipes",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://accounts.google.com/o/oauth2/token",
  "auth_provider_x509_cert_url":
  ↪  "https://www.googleapis.com/oauth2/v1/certs",
  "client_secret": "CLIENT_SECRET"
}
```

Listing 3: Simplified version of client_secret.json

The YouTube Data API is the REST API that helps in retrieving the data by sending an HTTP request to various endpoints with query parameters, which are supported by this API.

In order to get users' taste preferences in terms of likes from their YouTube profile, this API provides the *videos* endpoint, which is displayed below in the HTTP *GET* request. In this endpoint, various query parameters can be passed such as *part*, *chart*, *id*, *myRating*, *h1*, *maxHeight* or *maxResults* [You18c].

```
GET https://www.googleapis.com/youtube/v3/videos?{parameters}
```

The python code snippet depicted in Listing 4, creates an HTTP *GET* request to the *videos* endpoint, which provides the response body in a JSON format. It includes users' liked videos and other details. For our research, we are passing three query parameters, shown in Table 4.1.

A successful *200* HTTP *GET* request returns the JSON response, which comprises *thumbnails*, *title*, *defaultAudioLanguage*, *localized*, *channelId*, *publishedAt*, *description*, *title*, *tags*, *liveBroadcastContent* and *id*. This response provides a lot of information, which is not required for our research. Therefore, we filter this result and kept just the *description*, *title* and *id* of each liked video. The simplified JSON code snippet is depicted in Listing 5.

Query Parameters	Description
part	Set *snippet* specifies that the API response will include *channelId*, *title*, *description*, *tags*, and *categoryId* properties
myRating	Valid options are *like* and *dislike* set value = *like*, instruct the API to only return liked videos by the authenticated user
maxResults	Max value = *50* and Default value = *5* specifies the maximum number of items that the API response should return in the result set

Table 4.1: Query parameters of YouTube Data API to retrieve likes [You18c]

```
CLIENT_SECRETS_FILE = "client_secret.json"

# This OAuth 2.0 access scope allows for full read/write access to
# the authenticated user's account.
SCOPES = ['https://www.googleapis.com/auth/youtube.force-ssl']
API_SERVICE_NAME = 'youtube'
API_VERSION = 'v3'

def get_authenticated_service():
  flow =
  ↪  InstalledAppFlow.from_client_secrets_file(CLIENT_SECRETS_FILE,
  ↪  SCOPES)
  credentials = flow.run_console()
  return build(API_SERVICE_NAME, API_VERSION, credentials =
  ↪  credentials)

if __name__ == '__main__':
  client = get_authenticated_service()
  # calling function to retrieve users' likes by passing query
  ↪  parameters
  jsonResponse = videos_list_my_rated_videos(client, part='snippet',
  ↪  myRating='like',maxResults=50)
```

Listing 4: Code snippet of creating an HTTP *GET* request to *videos* endpoint

```
[
    {
        "title": "How To Make Tortilla Crusted Tilapia",
        "description": "FULL RECIPE BELOW\n\nThis tortilla crusted
        tilapia is a fun take on a traditional fried fish. Pair with
        a homemade salsa and tomatillo sauce. \n \nTORTILLA CRUSTED
        TILAPIA RECIPE: http://www.escoffieronline.com/....",
        "id": "hMgRhLl2DkI"
    },
    {
        "title": "Chipotle Bean Burrito Recipe",
        "description": "Get the Recipe:....", "....."
    },
    {
        "title": "Game of Thrones Season 8 Teaser Trailer #1
        (2019) Emilia Clarke, Kit Harington / Trailer Concept",
        "description": "Game of Thrones Season..","...."

    },
    {
        "title": "BEST QUINOA SALAD RECIPE EVER!
        (Colourful mint + turmeric salad)",
        "description": "OPEN ME!...","....."

    },{"....."}
]
```

Listing 5: Simplified version of YouTube JSON output

The full python source code for generating the above *GET* request and for filtering the JSON response is shown in Appendix A.1.

4.1.2 Facebook API

Similar to YouTube, Facebook also provides a REST API named Facebook Graph API, enabling third-party developers to access user's identity, photos, statuses, videos, likes and conversations [Arn18, Fac18a]. To retrieve this data, developers are required to generate an access token from Facebook developers tool and then send HTTP requests to various endpoints with query parameters supported by this API.

In order to get users' taste preferences in terms of likes from their Facebook profile, Facebook Graph API provides *likes* endpoint shown in HTTP *GET* request below.

```
GET https://graph.facebook.com/v2.12/{user-id}/likes
```

The python code snippet depicted in Listing 6, creates an HTTP *GET* request to *likes* endpoint that provides response body in JSON format and includes users' likes from their Facebook profile. Due to security reasons, snippet code does not contain *ACCESS_ TOKEN*.

```
# build the URL for the API endpoint
host = "https://graph.facebook.com"
path = "/{user-id}/likes"
ACCESS_TOKEN = "ACCESS_TOKEN"
params = urllib.urlencode({"access_token": ACCESS_TOKEN})

url = "{host}{path}?{params}".format(host=host, path=path,
↪  params=params)

# open the URL and read the response
resp = urllib.urlopen(url).read()
```

Listing 6: Code snippet of creating an HTTP *GET* request to *likes* endpoint

The JSON code snippet in Listing 7 is the simplified version of the successful 200 HTTP response, which includes *id*, *name* and *createdtime* of retrieved likes.

On the grounds of the JSON response shown in Listing 7, we observed that for each like, the information retrieved is insufficient. Hence, to perceive whether this information is sufficient for identifying the category of each like, we performed natural language text analysis on the *name* attribute using the AYLIEN Text Analysis API. The result obtained shows poor accuracy. To accurately identify the category, we need more information for each like, and therefore, we send an additional HTTP *GET* request to the endpoint of each like with two comma separated fields as query parameters shown below.

```
GET https://graph.facebook.com/v2.12/{id}?fields=description,
↪  about
```

```
{
    "data": [
        {
            "name": "Nachos",
            "id": "21755646521",
            "created_time": "2018-03-27T06:23:06+0000"
        },
        {
            "name": "The Amazing World of Gumball",
            "id": "142825982446439","..."
        },
        {
            "name": "Backstreet Boys",
            "id": "5736008378", "...."
        },
        {
            "name": "Lasagna",
            "id": "8562300962", "...."
        },{"......"}
    ]
}
```

Listing 7: Simplified version of Facebook JSON output

The two query parameter fields are [Fac18b]:

1. *description:* gives the detailed information about the like
2. *about:* gives brief information about the like

We send two fields instead of one because for each like either we are retrieving *description* or *about*. A successful *200* HTTP response returns the JSON output for each like. For an illustration, the JSON code snippet in Listing 8 depicts the response body retrieved for the *liked* page “Nachos", which has an *id* “21755646521".

```
    {
        "description": "We are a couple of individuals who enjoy the flavors
        ↪  of our favorite Nachos, So why not share ur love of nachos with
        ↪  everyone else....",
        "id": "21755646521"
    }
```

Listing 8: Simplified version of Facebook JSON output for a specific like

The full python source code in Appendix A.2 shows how we have created the HTTP *GET* request to retrieve users' likes, and then how an HTTP *GET* request is created to retrieve the additional information for each like.

4.2 Integrating NLP Techniques

This section presents the natural language processing techniques used for categorizing recipe likes from eclectic list of likes and for identifying the most relevant match for each recipe or ingredient retrieved from different sources.

4.2.1 AYLIEN Text Analysis API

The AYLIEN Text Analysis API is the REST API and follows natural language processing techniques for classifying text. For our research, we need to extract the recipe likes from users' eclectic list of likes retrieved from their YouTube and Facebook profiles. To achieve this, the *Text Analysis Component* of CAPRecipes (cf. Section 3.3.5) uses this API that analyses the information for each like and determines from which category these likes belong.

In order to use this API, developers need to create an account on AYLIEN Developer Portal for obtaining API ID and API Key. Although the service is paid, it is currently permitting 1000 free requests per day [Ayl18d].

Developers can use this API by sending an HTTP request to various endpoints with path and query parameters supported by the API.

For classifying the text, developers can use the *classification by taxonomy* endpoint of this API by passing path and query parameters in the HTTP *GET* request as shown below. The valid options for path parameter are *iab-qag* and *iptc-subjectcode*. The query parameters, which are supported by this API are depicted in Table 4.2.

```
GET https://api.aylien.com/api/v1/classify/:taxonomy?
↪ queryparameter=""
```

The python code snippet depicted in Listing 9 is for the text analysis of YouTube likes, which generates an above HTTP *GET* request to the *classification by taxonomy* endpoint with the path parameter as *iab-qag* and the query parameter as *text*.

Query Parameters	Description
text	*string* Text to classify
url	*string* Url to classify
language	*string en (Default)* Language

Table 4.2: Query parameters of AYLIEN Text Analysis API to identify the category of each like [Ayl18c]

```
# To identify recipe likes retrieved from the eclectic list of YouTube
↪  likes using Aylien Text Analysis API
def aylienTextAnalysis(jsonData):
  client = textapi.Client("API_ID", "API_KEY")
  recipeVideos = []
  for eachVideo in jsonData:
    # passing description attribute of each YouTube like as text query
    ↪  parameter and iab-qag taxonomy attribute as a path parameter
    classifications = client.ClassifyByTaxonomy({"text":
    ↪  eachVideo["description"].encode('utf-8'), "taxonomy":
    ↪  "iab-qag"})
    for category in classifications['categories']:
        if category["label"] == "Food & Drink":
            # Add as a recipe like if description belongs to Food and
            ↪  Drink category
            title = eachVideo["title"].encode('utf-8')
            recipeAttr = title.split("|")
            recipeVideos.append(recipeAttr[0].rstrip())
  return recipeVideos
```

Listing 9: Code snippet of creating an HTTP *GET* request to *classification by taxonomy* endpoint

In the python code snippet shown in Listing 9, we pass the *description* attribute of each YouTube like as the *text* query parameter. If the AYLIEN API analyses that the *description* belongs to the category of *Food and Drink*, then the *title* of that specific like will be added in the *youTubeRecipeLikes* JSON file that contains *titles*

of all recipe likes retrieved from YouTube for a given user. The developer API ID and API Key are not included due to security reasons. The JSON code snippet in Listing 10 depicts YouTube recipe likes determined by this API.

```
{
    "youtubeRecipeLikes": [
        "How To Make Tortilla Crusted Tilapia",
        "Chipotle Bean Burrito Recipe",
        "BEST QUINOA SALAD RECIPE EVER!  (Colourful mint + turmeric
        ↪  salad)",
        "Italian Sausage Lasagna",
        "........."
    ]
}
```

Listing 10: Simplified version of JSON output for YouTube recipe likes

In case of Facebook likes, Text Analysis, as explained above, for each like we retrieve either the *description* or *about* attribute. Therefore, we first identify whether the like contains the *description* or *about*. If it contains the *description* attribute, we pass it as the text query parameter for text analysis because the *description* renders detailed information of each *like*. Otherwise, we pass the *about* attribute as the text parameter. If the API determines that the like belongs to the *Food and Drink* category, the *name* of that specific like is added to the *facebookRecipeLikes* JSON file. This JSON file includes all the recipe likes gathered by exploiting users' Facebook profile, shown in the Listing 11.

```
{
    "facebookRecipeLikes": [
        "Lasagna",
        "Nachos",
        "Mexican Salad",
        "...."
    ]
}
```

Listing 11: Simplified version of JSON output for Facebook recipe likes

The full python source code of AYLIEN Text Analysis API for YouTube is shown in Appendix A.1 and for Facebook is in Appendix A.2.

4.2.2 POS Tagging Mean Probability Algorithm

This section explains the *POS Tagging Mean Probability* algorithm used by *Matching Component* of CAPRecipes (cf. Section 3.3.6) to find the most relevant match for recipes or ingredients having different names.

Part-of-Speech Tagging

The Parts Of Speech tagging (POS) is a principal component of the natural language processing analysis. It classifies words into their parts of speech and labels them accordingly such as noun, verb, adjective, adverb, and pronoun [Bog18]. In technical terms, POS tagging is the automatic assignment of descriptors, or tags, to input tokens, where the tags are the appropriate grammatical descriptors for words in text [ES15]. For labeling words in POS tagging, a tagset library is used, which is a collection of tags [SBL]. Currently, various tools and libraries are available for POS tagging, and this research is exploiting the most popular tagset of NLTK, known as *Penn Treebank* tagset. The key benefit of using this tagset is that the tags are well trained [Bog18]. A simple example of POS tagging is manifested in Listing 12.

```
from nltk import word_tokenize, pos_tag

print pos_tag(word_tokenize("I'm learning NLP"))
# [('I', 'PRP'), ("'m", 'VBP'), ('learning', 'VBG'), ('NLP', 'NNP')]
```

Listing 12: Code snippet of an example of Parts Of Speech tagging

Natural Language Toolkit

Natural Language Toolkit (NLTK) is a leading platform used for building Python programs and can work with the entire natural language processing methodology [Har18]. It contains more than 50 corpora and lexical resources such as WordNet, tokenization, stemming, tagging, parsing, or semantic reasoning. This library is used in various events like identifying named entities, displaying a parse tree and tokenizing and tagging some text [Nat18].

In our research, the POS Tagging technique applies an information retrieval method, which is a mean probability method that enables to perform a search for finding the most relevant match for each recipe or ingredient. Algorithm 1 reveals the steps we apply in the POS tagging technique to get the desired result and is called during the following events:

- **Similar Ingredients Matching:** To sort a recipes' list retrieve from the Spoonacular API by the sorted list of ingredients expiry, we need to compare its ingredients with recipes list's ingredients. As argued in Section 3.3.6, there might be a possibility that some ingredients have distinctive names in both lists. Thus, this algorithm assists in identifying similar ingredients with different names.

  For instance, the sorted ingredients expiry lists contain ingredients with names as *potato* and *cilantro*. And the recipes list contains recipes, which have ingredients with names as *reddish-brown potato* and *fresh cilantro*. As we can see, both lists have same ingredients with different names, that can be easily identified by *POS Tagging Mean Probability* algorithm.

- **Similar Recipes Matching:** Similar to ingredients matching, this algorithm matches similar recipes with different names and utilizes in subsequent situations:

  1. To incorporate the recipes corresponding to the users' taste preferences retrieved either from *YouTube*, *Facebook* or *Similar Users* into the users' personal dynamic context, we first compare recipes enclosed in users' personal dynamic context with the retrieved recipes to eliminate duplication.
  2. To incorporate users' taste preferences in recipe recommendations, we sort and refine the recipes retrieved from the Spoonacular API with users' taste

preferences (cf. Section 5.2). For an illustration, the flow diagram depicted below explains the steps of this algorithm, particularly for this scenario.

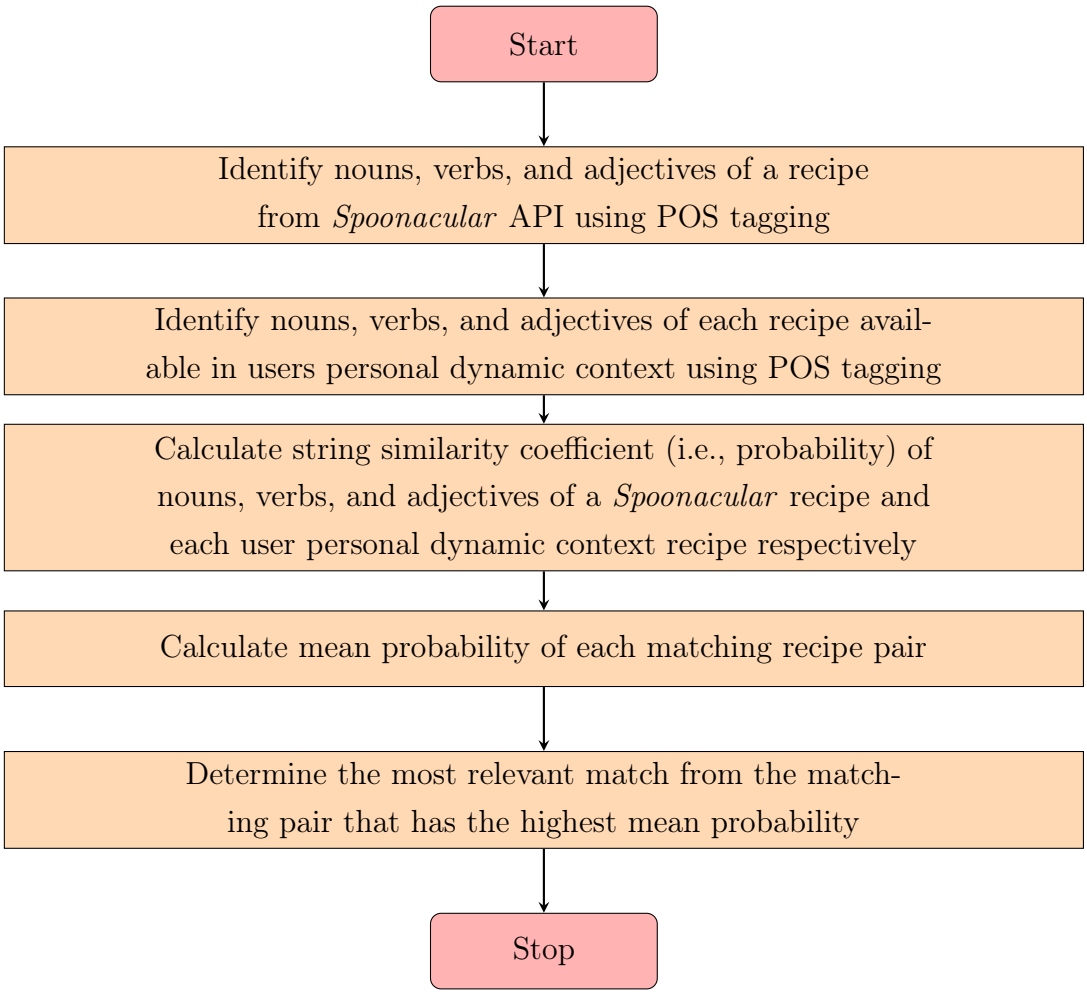


Table 4.3 shows the most relevant matched recipe name from the users' personal dynamic context after applying the POS Tagging Mean Probability algorithm on some of the Spoonacular recipes.

Recipes from Users' Taste Preferences	**Spoonacular Recipes**
Quinoa Enchilada Casserole	Quinoa Enchilada Casserole
How To Make Tortilla Crusted Tilapia	Tortilla-Crusted Tilapia with Mango Salsa
Chipotle Bean Burrito Recipe	Bean Burritos
Mexican Tuna Tacos	Grilled Tuna with salsa
BEST QUINOA SALAD RECIPE EVER! (Colourful mint + turmeric salad)	Quinoa Salad
Italian Sausage Lasagna	Lasagna With Sausage Ragu

Table 4.3: Some of the most relevant matches of recipes

Algorithm 1 POS Tagging Mean Probability Algorithm

for each *recipe* in recipeList **do**
  Set $P(N) = 0$ ▷*Set initial noun(N) probability to zero*
  Set $P(V) = 0$ ▷*Set initial verb(V) probability to zero*
  Set $P(A) = 0$ ▷*Set initial adjective(A) probability to zero*
  Set $P(X) = 0$ ▷*Set mean probability to zero*
  Set $counter = 0$
  Set $most\_relevant\_recipe = str()$
  Set $matching\_threshold = 0.7$ ▷*Set the minimum probability for matching nouns, verbs, and adjectives to 0.7*
  Set $max\_mean\_probability = 0.7$ ▷*Set the maximum mean probability to 0.7*
  Extract the sets of nouns *N1*, verbs *V1*, and adjectives *A1* using POS tagging of a *recipe*
  Retrieved list of recipes for matching
  for each *rec* from the retrieved recipes list **do**
    Extract the sets of nouns *N2*, verbs *V2*, and adjectives *A2* using POS tagging of a *rec*
    Calculate the string similarity coefficient as the probability between *N1*, *V1*, *A1* and *N2*, *V2*, *A2* respectively
    if $noun\_probability > matching\_threshold$ **then**
      $P(N) += noun\_probability$
    end if
    if $verb\_probability > matching\_threshold$ **then**
      $P(V) += verb\_probability$
    end if
    if $adjective\_probability > matching\_threshold$ **then**
      $P(A) += adjective\_probability$
    end if
    $P(X) = (P(N) + P(V) + P(A))/3$ ▷*Calculate mean probability*

```
        if P(X) > max_mean_probability then
            max_mean_probability = P(X)
            most_relevant_recipe = rec
            counter+ = 1
        end if
    end for
    return the most relevant match, most_relevant_recipe
end for
```

Note that in the algorithm 1, we set the maximum mean probability threshold to 0.7 because below this threshold, the matching results achieved were very poor.

The full python source code for finding the most relevant match for recipes or ingredients using POS Tagging Mean Probability algorithm is shown in Appendix A.3

4.3 Integrating Collaborative Filtering

"A lot of times, people don't know what they want until you show it to them [Aro18]" - *Steve Jobs.*

The taste preferences retrieved from above techniques are users' own preferences, which they have liked on various platforms i.e., YouTube and Facebook. We use collaborative filtering in our recommendation engine to better understand users' taste preferences as it helps in predicting the preferences for recipes, which users have not liked yet.

Collaborative filtering helps in predicting the user's preferences by compiling preferences from several other users in light of their similarities. Similarities in users are identified by viewing at their overlap in preferences for the items, which is known as the *user-based* method [Sid]. Another method of this technique is *item-based*, which looks into a set of items that the user has already liked or rated and predicts user preferences by compiling preferences from several other items in view of similarities. With this method, instead of computing the similarity between two users, we focus on the similarity between two items [LZW13]. The high-level overview of both the methods are depicted in Figure 4.1.

In recent years, adoption of collaborative filtering has increased [TB14] as it is proven to be one of the most successful techniques to build personalized recommendation systems [LZW13, BK11, Tec18b]. This technique is widely used in social

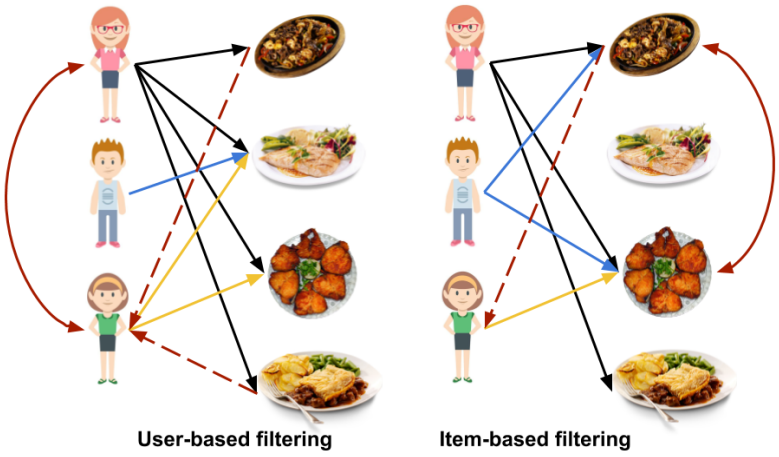


Figure 4.1: High level overview of User-based and Item-based methods [Mar18]

media, e-commerce, or streaming services such as Facebook,[1] YouTube,[2] Amazon,[3] Netflix,[4] iTunes,[5] IMDB,[6] LastFM,[7] Delicious[8] and StumbleUpon[9] [TB14, Tec18b].

For instance, Harshit watched and liked several videos on Facebook. As Facebook uses collaborative filtering, it can look at other users who liked those same videos and recommend ones which other similar users liked, but Harshit might not have seen yet [Kee18].

Algorithm 2 exemplifies the user-based collaborative filtering technique used by the recommendation engine of CAPRecipes for generating a recipe preferences list for a user by computing the similarity with other users. To achieve this, we forward users' personal dynamic context to this algorithm, which provides a result in JSON format, the simplified version is shown in the Listing 13. In this method, we first find a set of the nearest neighbor users for a user whose past likes on recipes have a robust correlation. Then, this algorithm predicts the scores for unobserved recipes depending on the combination of scores retrieved from the nearest neighbors [BS97]. The fundamental part of this technique is computing similarity between users, and we have several popular measures for it such as *Cosine Similarity*, *Spearman Correlation*,

[1] https://www.facebook.com
[2] https://www.youtube.com
[3] https://www.amazon.com
[4] https://www.netflix.com
[5] https://www.apple.com/ca/itunes
[6] http://www.imdb.com
[7] https://www.last.fm
[8] https://del.icio.us
[9] https://www.stumbleupon.com

and *Pearson Correlation* [Eks18, WDVR06]. We use *Pearson Correlation* because it has been recognized that it provides better accuracy than other measures [WDVR06]. Also, user-based collaborative filtering is the method known for its simplicity and efficiency [BK11].

```
{
    "collaborativeRecipeLikes": [
        "Grilled Fish Tacos",
        "Broiled Shrimp Scampi"
        "Seductive Salmon With Rainbow Salsa",
        "...."
    ]
}
```

Listing 13: Simplified version of recommended recipes by Collaborative filtering

For an illustration, Table 4.4 depicts four users, six recipes and their preferences on it, and in this scenario, we are predicting the preferences for a user, Harshit, which are retrieved by computing similarity using the Algorithm 2. This algorithm computes that Harshit is most similar to Logan, and suggests the recipes' preferences from the Logan's preferences list in which Harshit may show interest. The suggested recipes' preferences are *Grilled Fish Tacos* and *Seductive Salmon with Rainbow Salsa* as highlighted with red in Table 4.4.

	Hot Nicoise Salad	**Grilled Fish Tacos**	**Arugula Lasagna**	**Grilled Sea with Tropical Salsa**	**Seductive Salmon with Rainbow Salsa**	**Grilled Tuna with salsa**
Harshit		👍	👍		👍	👍
Alice	👍		👍	👍		
Logan		👍	👍		👍	👍
Bob		👍		👍		👍

Table 4.4: Suggested recipes' preferences to Harshit by Collaborative filtering

Algorithm 2 User-Based Collaborative Filtering Algorithm for User A

```
Set totals = dict()
Set simSums = dict()
Set recommendedList = list()
Set rankings = list()
Set counter = 0
Set sim_threshold = 0.5                ▷Setting the minimum threshold value of similarity to 0.5
dataset = Query Personal Dynamic Context of All Users
for each user in dataset do
    if user ≠ userA then
        calling pearson correlation function to calculate similarity between users,
        sim = pearson_correlation(userA, user)
    end if
    if sim > sim_threshold then
        for each item in dataset[user] do
            if item is not rated by user A then
                totals[item] = 0
                totals[item] = totals[item] + dataset[user][item] * sim   ▷Calculating average rating
                for each item based on other users rating and the similarities
                simSums[item] = 0
                simSums[item] = simSums[item] + sim                       ▷Sum of similarities
            end if
        end for
    end if
end for
for each item, total in totals do
    rankings[counter] = (total/simSums[item], item)                       ▷Create the normalized list
    counter+ = 1
end for
for each score, rec_item in rankings do
    if score>=0.5 then                 ▷Return recipes that have average rating >= 0.5
        recommendedList.append(rec_item)
    end if
end for
return the recommended list, recommendedList
```

Note that in the algorithm 2, the minimum similarity threshold is set to 0.5 to get the better prediction quality of the missing data.

4.4 Integrating Dynamic Users' Preferences

Once we retrieve the recipes' preferences by exploiting various sources using the services and algorithms explained in the previous sections, we need to concatenate them in the users' personal context, which has the JSON format. As we are gathering the preferences from various sources, it is possible that these sources have likes of similar recipes with distinctive names. Therefore, to eliminate duplicates, we are passing these preferences to the POS Tagging Mean Probability algorithm (cf. Section 4.2.2) before concatenating the preferences into the users' personal dynamic context. For instance, the retrieved recipes' preferences from YouTube having name *Italian Sausage Lasagna* and from users' personal dynamic context *Lasagna With Sausage Ragu*, are similar recipe with different names, then the concatenation process applied by this algorithm keeps the one retrieved from users' personal dynamic context. Figure 4.2, manifests the high-level overview of the concatenation process.

Algorithm 3 illustrates the process of concatenating the taste preferences into the users' personal dynamic context. This algorithm is called in situations such as:

- *Integrating YouTube Preferences*
- *Integrating Facebook Preferences*
- *Integrating Similar User Preferences*

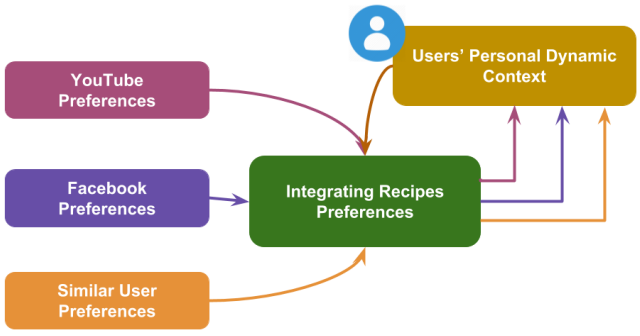


Figure 4.2: High level overview of the concatenation process

Algorithm 3 Algorithm to Integrate Taste Preferences into Users' Personal Context

1: **function** IntegratingPreferences(*recipe_list*, *per_dyn_context*)
2: **for** each *recipe* name in *recipe_list* **do**

```
3:          find match in per_dyn_context, match = pos_tag(recipe, per_dyn_context)  ▷Calling
            the POS tagging script to find the most relevant match of recipes
4:          if match then
5:              per_dyn_context.append(recipe)
6:          end if
7:      end for
8:      return per_dyn_context
9:  end function
10: function MAIN()
11:     Set combined_likes = list()
12:     Extract youtube_likes from user's YouTube JSON file
13:     Extract facebook_likes from user's Facebook JSON file
14:     Extract collaborative_likes from user's Collaboartive JSON file
15:     Extract per_dyn_context from user's Personal Context JSON file
16:     combined_likes = youtube_likes + facebook_likes + collaborative_likes
17:     per_dyn_context = IntegratingPreferences(combined_likes, per_dyn_context) ▷Find
        and update the most relevant match of recipe in personal dynamic context
18:     return per_dyn_context
19: end function
```

The full python source code of integrating preferences retrieved from various sources into the users' personal dynamic context is shown in Appendix A.4.

4.5 Summary

This chapter presented the services and algorithms used to extract users' preferences. We first explained how to exploit third-party social media apps, and then apply collaborative filtering, which computes the similarity matrix for a given user. We discussed the AYLIEN Text Analysis API that is utilized for identifying users' recipe preferences from the list of users' preferences retrieved from their social media profiles. Moreover, we demonstrated POS tagging technique that efficiently addresses the issues of having multiple names of same recipe or ingredient by finding the most relevant match for each recipe or ingredient. We then explained how we are integrating all the users' taste preferences retrieved from various sources that help in providing relevant recipe recommendations. This chapter is a key contribution to our research questions 1 and 2 (cf. Section 1.1).

Chapter 5

Exploiting Users' Context for Personalized Recommendations

This chapter illustrates the service and algorithm that exploit users' kitchen, personal and temporal context for providing personalized recipe recommendations. This chapter contributes to our research questions 3, 4 and 5.

5.1 Integrating Recipe Service

This section presents the recipe service which is used as a recipe database in the recommendation engine of CAPRecipes.

5.1.1 Spoonacular API

The Spoonacular API is the Recipe, Nutrition and Food API that has more than 365K recipes as data. It is a REST API that offers numerous features and is recognized as one of the best APIs for recipes [JL18]. Therefore, the *Recipe Component* of CAPRecipes uses this API as a recipe database instead of creating one.

To access this API, developers need an API key, which can be generated either through the Mashape or Rapid platforms. Although the service is paid, it is currently permitting 50 free requests per day, however for hackathons and academic purposes, developers can get access at a meager price of $10 for 5000 requests per day [Spo18].

The Spoonacular API provides various endpoints that enable developers to incorporate numerous recipe related functionality into their applications. This API supports several of the most popular programming languages like Java, PHP, Python,

Objective-C, Ruby, and, .Net to create an HTTP request to the endpoint with query parameters, and returns the response body in JSON format. For our research, we are using *searchComplex* endpoint of this API, illustrated below

```
GET https://spoonacular-recipe-food-nutrition-v1.p.mashape.com
↪ /recipes/searchComplex?{queryParameters}
```

The *searchComplex* endpoint provides various query parameters which can be used by the developers as per the requirements of their application. Table 5.1 shows the query parameters we are consolidating in our research.

Query Parameters	**Description**
cuisine	*string* one or more (comma separated) cuisines, such as: african, chinese, japanese, korean, vietnamese, thai, indian, british, irish or french
diet	*string* diet to which the recipes must be compliant, such as: pescetarian, vegetarian or vegan
intolerances	*string* one or more (comma separated) allergies, such as: egg, shellfish, soy or wheat
addRecipeInformation	*boolean* If set to true, get more information about the recipes
fillIngredients	*boolean* add information about the used and missing ingredients in each recipe
maxCalories	*number* maximum number of calories the recipe can have
maxCarbs	*number* maximum number of grams of carbohydrates the recipe can have

maxFat	*number* maximum number of grams of fat the recipe can have
maxProtein	*number* maximum number of grams of protein the recipe can have
type	*string* type of recipes returned, such as main course or lunch
number	*number* number of results to return (Min value = 1 and Max value = 100)

Table 5.1: Query parameters of the Spoonacular API to retrieve recipes [Mas18]

```
# calling Spoonacular API to retrieve recipes
apicall =
 ↪  "https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes
 ↪  /searchComplex?addRecipeInformation=true&cuisine="+cuisineinfo+"&
 ↪  diet="+diet+"&excludeIngredients=&fillIngredients=true&
 ↪  includeIngredients="+ingredientsString+"&intolerances="+allergies+"
 ↪  &limitLicense=false&maxCalories="+maxCalories+"&maxCarbs="+maxCarbs+"
 ↪  &maxFat="+maxFat+"&maxProtein="+maxProteins+"&number="+recipeCount+"
 ↪  &offset=0&ranking=2&type="+dishtype

response = unirest.get(apicall,
headers={
"X-Mashape-Key": "Developer_API_Key",
"Accept": "application/json"
}
)
```

Listing 14: Code snippet of creating an HTTP *GET* request to *searchComplex* endpoint

The python snippet code of the API call depicted in Listing 14 creates an HTTP *GET* request to the *searchComplex* endpoint with the query parameters shown in Table 5.1. The developer API key is not included in snippet code due to security

reasons. The query parameter we are forwarding to this endpoint is the users' personal static, kitchen and temporal context.

A successful 200 HTTP request returns a JSON response. For an illustration, Listing 15 is the simplified version of JSON response, which shows the list of recipes with the requested information of each recipe obtained for a given user based on her personal, kitchen and temporal context.

```
[{
    "carbs": "48g",
    "title": "Tortilla-Crusted Tilapia with Mango Salsa",
    "cuisines": ["mexican"],
    "image": "https://spoonacular.com/recipeImages/2714-312x231.jpg",
    "calories": 898,
    "fat": "38g",
    "readyInMinutes": 45,
    "usedIngredients": [
        {
            "name": "cilantro",
            "image": "https://spoonacular.com/cdn/ingredients_100x100
            ↪  /cilantro.png",
            "amount": 0.33,
            "..."
        },{"..."}],
    "missingIngredients": [{"name": "lime juice","...."},{"..."}],
    "protein": "93g",
    "cookingInstructions": [
        {
            "equipment": [
                {
                    "name": "frying pan"
                },{"..."}],
            "step": "Place fish in the pan and cook...",
            "ingredients": [{"name": "fish"},{"..."}]
        }]
},
{
    "carbs": "41g",
    "title": "Grilled Fish Tacos",
    "fat": "73g", "..."
},{"....."}]
```

Listing 15: Simplified version of the Spoonacular JSON output

5.2 Personalized Recipes Engine Algorithm

There are several techniques for designing personalized recommendation systems. The three most important techniques utilized for building these systems are content-based, collaborative and hybrid-based filtering. It has been discovered from recent research that hybrid-based filtering is more efficient and provides greater precision. Hybrid-based filtering is a combination of content-based and collaborative filtering. There are various ways to implement this technique, such as separately making predictions from content-based and collaborative and then combine them, adding the capabilities of content-based filtering into collaborative filtering and vice versa, or both filtering can be unified into one model. Netflix is one of the world's most admired companies [Lau18], and they also use hybrid-based filtering to recommend movies to users. They use collaborative filtering to match the watching and searching habits of similar users, and use content-based filtering to offer movies that share characteristics with movies that a user has highly rated [Rec18].

Our research also exploits hybrid-based filtering by unifying content-based and collaborative filtering techniques into the CAPRecipes recommendation engine (cf. Figure 3.13). Moreover, to improve our recommendations, we are also exploiting various natural language processing techniques such as text analysis and pos tagging (cf. Section 4.2).

The JSON response retrieved from the Spoonacular API uses content-based filtering, which internally uses natural language processing to provide recipes according to the users' health restrictions and nutritional goals and availability of ingredients. As explained in Chapter 1 and 2, users' taste preference is a key factor while choosing recipes. Additionally, users' taste preferences and ingredients' expiry influence the problem of food wastage. Therefore, the Algorithm 4 (i.e., *Personalized Recipes Engine Algorithm*) further refine and sort the recipes list retrieved from the Spoonacular API for incorporating users' taste preferences and expiry of ingredients in recipe recommendations. In the CAPRecipes recommendation engine, the CAPRecipes *Recommender Component* utilizes this algorithm (cf. Section 3.3.9).

In this algorithm, three lists in the JSON format, are forwarded as an input, which are orchestrated by leveraging various components of CAPRecipes. These lists are:

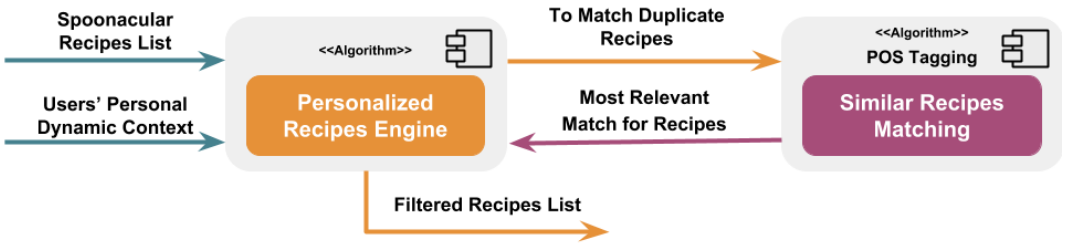
1. Recipes list obtained from the Spoonacular API

2. Users' kitchen context

3. Users' personal dynamic context

Based on these three lists, the Algorithm 4 further sort and refine the *Spoonacular recipes list*, which is illustrated with the following steps:

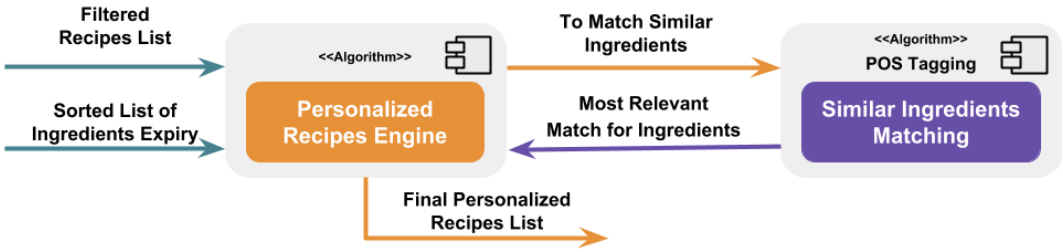
- For incorporating users' taste preferences to make the recipes list more personalized, this algorithm initially compares the *Spoonacular recipes list* against *users' personal dynamic context* and gives a *filtered recipes list* in JSON format with the aid of the POS Tagging Mean Probability algorithm (cf. Section 4.2.2).



- Then, to sort the above *filtered recipes list*, containing the maximum number of nearly expiring ingredients that help in promoting the reduction of food wastage, this algorithm:

  1. Firstly computes the *sorted list of ingredients* in JSON format based on its expiry dates by exploiting users' kitchen context.

  2. Then, the *filtered recipes list* is further filtered with the *sorted list of ingredients expiry* with the help of POS Tagging Mean Probability algorithm, to provide the *final personalized recipes list* in JSON format, which displays to users as CAPRecipes personalized recipe recommendations.



Algorithm 4 Algorithm to Recommend Personalized Recipes List

function REFININGWITHTASTE($sp\_recipes\_list, recipe\_preferences$)
  Set $match\_recipes\_list = list()$
  Set $unmatch\_recipes\_list = list()$
  Set $taste\_pref\_list = list()$
  for each *recipe* name in $sp\_recipes\_list$ **do**
    find *match* in $sp\_recipes\_list$, $match = pos\_tag(recipe, recipe\_preferences)$ ▷*Calling the POS tagging script to find the most relevant match of recipes*
    if *match* **then**
      $match\_recipes\_list.append(recipe)$
    else
      $unmatch\_recipes\_list.append(recipe)$
    end if
  end for
  $taste\_pref\_list = match\_recipes\_list + unmatch\_recipes\_list$
  return $taste\_pref\_list$
end function

function REFININGWITHEXPIRY($taste\_rec\_list, sr\_ingre\_list$)
  Set $expiry\_recipe\_list = list()$
  for each *recipe* in $taste\_rec\_list$ **do**
    Set $date =$ ""
    Set $counter = 0$
    for each $used\_ingredient$ in *recipe* **do**
      find *exp* in sr_ingre_list, $exp = pos\_tag(used\_ingredient, sr\_ingre\_list)$ ▷*Calling the POS tagging script for matching the ingredients of users' kitchen context with Spoonacular ingredients*
      if *exp* **then**
        $counter+ = 1$
        if $date ==$ ""$or\ date > exp$ **then**
          $date = exp$
        end if
      end if

```
        end for
        if counter == 0 then
            currentDate = today()                          ▷Setting current date with todays date
            lastAssignedDate = currentDate + timeDelta(days = 11)  ▷Calculating last assigned
            expiry date by adding 11 days delay to the current date
            date = lastAssignedDate
        end if
        recipe[exp] = date
        recipe[exp_ingredients] = counter
    end for
    expiry_recipe_list.append(recipe)
    Sort expiry_recipe_list based on expiry date in ascending order
    Sort expiry_recipe_list in descending order based on expiring ingredients numbers
    return expiry_recipe_list
end function

function MAIN()
    Set sorted_ingre_list = list()
    Set spoon_recipes_list = list()
    Set taste_ref_recipes_list = list()
    Set final_recipes_list = list()
    Extract pers_dyn_context and kitchen_context from user's JSON file
    spoon_recipes_list = Query Spoonacular API using kitchen, personal static and temporal -
    context
    sorted_ingre_list = SortingIngredientsWithExpiry(kitchen_context)      ▷Calling the
    script to sort the ingredients
    taste_ref_recipes_list = RefiningWithTaste (spoon_recipes_list, pers_dyn_context)
    final_recipes_list = RefiningWithExpiry (taste_ref_recipes_list, sorted_ingre_list)
    return final_recipes_list
end function
```

The full python source code for generating an HTTP *GET* request to the Spoonacular API and to further refine and sort recipes by exploiting users' taste preferences and ingredients expiry is shown in Appendix A.5.

5.3 Summary

This chapter illustrated the Spoonacular API service that tackles user's health-related issues and poor availability of ingredients by providing recipe recommendations according to a user's health restrictions, their nutritional goals and maximizing the use

of ingredients available in their kitchen. Then we discussed the *Personalized Recipes Engine* algorithm which is a *hybrid-based* filtering algorithm that unifies *content-based* and *collaborative* filtering algorithms along with *NLP* techniques. This algorithm is utilized to further sort and refine the recipes list retrieved from the Spoonacular API to incorporate user taste preferences and expiry of ingredients in recipe recommendations. This chapter contributed to our last three research questions (cf. Section 1.1).

Chapter 6

Results, Analysis and Evaluation

This chapter discusses the overview of evaluation in which we demonstrate the three quality factors of usability to evaluate the prototype of CAPRecipes using two experiments. We also illustrate both experiments that include scenarios in which we explain considerations users face during the selection of recipes, their context as sensed by CAPRecipes, and the results retrieved based on their considerations. In addition, we perform analysis on the results and evaluate the level of fulfillment of the intended goal by CAPRecipes for both experiments.

6.1 Evaluation Approach

The aim of CAPRecipes is to help users improve their experience by providing personalized recommendations of recipes with minimal manual effort. Figure 3.9 shows the CAPRecipes web interface that comprises four parameters in the form of toggle buttons which provide users full control over personalization. By default, all parameters of CAPRecipes are enabled.

The implementation of CAPRecipes prototype is considered to be the part of evaluation. Moreover, CAPRecipes is evaluated with the help of two experiments while assessing the three quality factors of usability. The quality factors we used for the evaluation of CAPRecipes are defined by *ISO 9241-11* and are explained in the next section. However, we did not perform formal user studies.

The two volunteers we have chosen for the experiments were useful because they could challenge our system as they had complicated health restrictions and numerous considerations while selecting a recipe. However, for the experiment both decided to

first disable all the parameters and then enable each parameter one by one to identify how their personalized recommendations are enhanced gradually, and we had recorded the result for each parameter to evaluate the usability of CAPRecipes efficiently. Moreover, both volunteers are regular users of YouTube and Facebook and agreed to let CAPRecipes exploit their profiles to extract their taste preferences for providing more personalized recipe recommendations.

Note that, to protect the privacy of each volunteer the names for the experiments are changed.

6.1.1 Quality Factors of Usability

Usability: is defined as the level to which a product is utilized by particular users to accomplish specific objectives in a particular context of use [Bev95].

ISO 9241-11 states that usability can be evaluated via user performance and user satisfaction where user performance is defined in terms of two quality factors, which are effectiveness and efficiency. Hence, the three quality factors of usability are listed next [Bev95]:

1. **Effectiveness** is defined as the level to which the intended goals of use are achieved. Therefore, to evaluate the effectiveness of CAPRecipes we qualitatively analyzed the accuracy and personalization of the results retrieved with the intended goal. The intended goal is illustrated in the experiment of the first volunteer. In addition, for identifying effectiveness, we classified the accuracy of the results into three categories: barely, moderately, and highly.

2. **Efficiency** is defined as the resources, such as time or mental effort expended, to achieve the intended goals. We evaluate the efficiency of CAPRecipes by identifying the number of steps required by the volunteers to retrieve the preferred personalized recipes, taking into account their considerations.

3. **User Satisfaction** is defined as the level to which users find the use of the product acceptable. For a better understanding of the level of user experience felt by the volunteers, and the level of objective achieved by CAPRecipes, we formulated several quick questions which we asked the volunteers after the completion of each experiment. Also, we then classified user satisfaction into three categories: dissatisfied, moderately satisfied and highly satisfied.

6.2 Experiment 1

Steve is our first volunteer who is 32 years old and works for an IT firm in Victoria. He is a diligent employee who leaves for work at 8:00 am in the morning and returns home at 6:30 pm in the evening. He is a food lover and likes to cook his meals. Though there are some considerations which limit his meal options while selecting a recipe: First, he is suffering from obesity and is allergic to *peanut* and *broccoli*. Second, he has certain taste preferences which he does not want to compromise. Third, he prefers to cook a recipe by maximizing the use of ingredients available in his kitchen. Fourth, due to his obesity, his specialist asked him to follow the paleo[1] diet. Fifth, his preferable cuisines are Italian and Spanish. Lastly, he is aware of the food crises around the world, therefore aims to not waste food. However, he is tired of manually searching for healthy recipes, and having to apply filters in light of his above considerations every time.

Therefore, Steve hopes to use CAPRecipes to leverage his context while exploiting his numerous considerations to enhance his circumstances.

On March 23, 2018, at 7:30 pm in the evening, for this experiment, Steve chose a recipe to cook for dinner with the assistance of CAPRecipes. CAPRecipes sensed his current contexts and provided personalized recipe recommendations for Steve as outlined below:

[1]The Paleolithic diet consists mainly of fish, grass-fed pasture raised meats, eggs, vegetables, fruit, fungi, roots, and nuts, and excludes grains, legumes, dairy products, potatoes, refined salt, refined sugar and processed oils [Chr18a]

6.2.1 Context

Kitchen Context

It includes the ingredients available in Steve's kitchen with their expiry dates.

Ingredients	Expiry
carrot	14 days
veal shank	12 days
dry red wine	-
lamb	5 days
capers	14 days
onion	12 days
tomato	12 days

(a)

Ingredients	Expiry
chicken breast	3 days
garlic	15 days
shrimp	9 days
sherry wine	-
chicken broth	60 days
red wine vinegar	-
olives	60 days

(b)

Ingredients	Expiry
eggs	15 days
chorizo sausage	12 days
red snapper	6 days
serrano ham	1 days
shallot	5 days
chicken stock	25 days
red bell pepper	14 days
white wine	-

(c)

Table 6.1: Steve's kitchen context

Note that, for the experiment, it is assumed that herbs and seasoning used in Italian and Spanish cuisines are available in Steve's kitchen and have a long shelf life, and are therefore not included in Table 6.1.

Personal Static Context

Steve wants to be healthier, live longer, and avoid developing diseases associated with obesity. His specialist recommended him to follow the paleo diet and not exceed certain nutritional values per day in his meals. The following contexts depict his recommended nutritional values and diet, the ingredients he is allergic to, and the cuisines he prefers.

- *Allergy(ies):* Peanut and Broccoli
- *Nutritional Goals/day:* Calories-1500, Protein-120gm, Carbs-70gm, Fat-40gm
- *Diet:* Paleo
- *Cuisine(s):* Italian and Spanish

The current prototype of CAPRecipes is calculating per meal nutritional goals with the following logic:

$$NutritionalGoalsPerMeal = \frac{NutritionalGoalsPerDay}{3}$$

Steve's nutritional goals per meal were: Calories-500, Protein-40gm, Carbs-23gm, Fat-13gm.

Personal Dynamic Context

It includes Steve's liked recipes' preferences extracted by exploiting his Facebook and YouTube social media profiles and includes recipes' preferences suggested by collaborative filtering.

- Facebook Preferences: *Easy Lemon Pepper Scampi*, *Puttanesca Baked Cod*, *Spanish Lamb Chops*, *Salmon And Scallop Skewers With Romesco*, *Italian Chicken and Artichokes*
- YouTube Preferences: *Red Snapper Puttanesca*, *Quick Braised Red Snapper Puttanesca with Pasta*, *Fish and Zucchini Puttanesca Stew*, *Spanish Pea Soup with Crispy Ham*, *Flavorful Chicken for Empanadas*, *Zesty Italian Chicken Salad*, *Chicken Cacciatore Ii*, *Arugula Pumpkin Seed Pesto Sauerkraut Lasagna*
- Similar Users Preferences: *Italian Fish Bake*, *Filetto Di Pomodoro*, *Broiled Shrimp Scampi*, *Cauliflower "risotto" With Fresh Basil (grain-free And Dairy-free)*, *Serrano ham, olive & parsely salad*

Personal Dynamic Context

Easy Lemon Pepper Scampi, *Puttanesca Baked Cod*, *Spanish Lamb Chops*, *Salmon And Scallop Skewers With Romesco*, *Italian Chicken and Artichokes*, *Red Snapper Puttanesca*, *Quick Braised Red Snapper Puttanesca with Pasta*, *Fish and Zucchini Puttanesca Stew*, *Spanish Pea Soup with Crispy Ham*, *Flavorful Chicken for Empanadas*, *Zesty Italian Chicken Salad*, *Chicken Cacciatore Ii*, *Arugula Pumpkin Seed Pesto Sauerkraut Lasagna*, *Italian Fish Bake*, *Filetto Di Pomodoro*, *Broiled Shrimp Scampi*, *Cauliflower "risotto" With Fresh Basil (grain-free And Dairy-free)*, *Serrano ham, olive & parsely salad*

Temporal Context

It includes the time, sensed by CAPRecipes, at which Steve requested recipe recommendations.

- 7:30 PM

6.2.2 Results & Analysis

In this section, we have shown the results retrieved for Steve when he had enabled each parameter one by one. Moreover, the ingredients that are highlighted in red are the missing ingredients of each recipe, and the ingredients that are highlighted in blue are Steve's allergic ingredients.

Based on Ingredients

Context Exploited

- Kitchen Context (Partial) i.e., name
- Personal Static Context (Partial) i.e., diet and preferable cuisines
- Temporal Context

Table 6.2 shows the top five recipes results with its necessary information, retrieved when Steve enabled only the *Ingredients* parameter.

S.No	I-R1	I-R2	I-R3	I-R4	I-R5
Recipes	 Lamb Osso Buco With Creamy Polenta	 Spanish Lamb Shanks Jerez	 Quick Braised Red Snapper Puttanesca	 Paella on the Grill	Spanish Tortilla with Broccoli, Chorizo, and Onion
Cuisine	Italian	Spanish	Italian	Spanish	Spanish
Diet	Paleo	Paleo	Paleo	Paleo	Paleo
Nutritional Info	Calorie:**1039** Protein:39g Carbs:22g Fat:**75g**	Calorie:416 Protein:37g Carbs:16g Fat:12g	Calorie:352 Protein:36g Carbs:7g Fat:13g	Calorie:461 Protein:39g Carbs:8g Fat:12g	Calorie:**569** Protein:15g Carbs:9g Fat:41g
Used & Missing Ingredients	celery chicken stock garlic Lamb thyme tomato paste yellow onion rosemary dry red wine carrots	bay leaves garlic cloves whole ground cloves lamb stew meat onions parsley stock sherry wine carrots	basil leaves capers garlic kalamata olives white skinless red snapper diced tomatoes red pepper flakes	chicken broth chorizo sausage curry powder garlic greek olives ground turmeric plum tomatoes sherry wine vinegar shrimp chicken breasts artichoke hearts	red wine vinegar diced chorizo eggs onion broccoli florets

Table 6.2: Steve's results when only Ingredients parameter was enabled

The recipes retrieved, i.e., *I-R1* to *I-R5* were listed in the descending order of

available ingredients. Moreover, the recipe *I-R5* contains the ingredient to which Steve is allergic to, i.e., Broccoli. In addition, the recipes *I-R1* and *I-R5* violates his nutritional goals as these two recipes have higher nutritional values than allowed.

Based on Ingredients and Health

Context Exploited

- Kitchen Context (Partial) i.e., name
- Personal Static Context
- Temporal Context

Once Steve retrieved the results by enabling only the *Ingredients* parameter, he then enabled the *Health* parameter to get recipe recommendations which do not violate his health restrictions and nutritional goals. In the Table 6.3, we have shown the results of the top five recipes (i.e., *H-R1* to *H-R5*), which he was presented with by CAPRecipes.

S.No	H-R1	H-R2	H-R3	H-R4	H-R5
Recipes	 Spanish Lamb Shanks Jerez	 Quick Braised Red Snapper Puttanesca	 Osso Buco with Red Wine	 Paella on the Grill	 Slow-Braised Osso Buco
Cuisine	Spanish	Italian	Italian	Spanish	Italian
Diet	Paleo	Paleo	Paleo	Paleo	Paleo
Nutritional Info	Calorie:416 Protein:37g Carbs:16g Fat:12g	Calorie:352 Protein:36g Carbs:7g Fat:13g	Calorie:364 Protein:40g Carbs:13g Fat:7g	Calorie:461 Protein:39g Carbs:8g Fat:12g	Calorie:364 Protein:37g Carbs:13g Fat:8g
Used & Missing Ingredients	bay leaves garlic cloves whole ground cloves lamb stew meat onions parsley stock sherry wine carrots	basil leaves capers garlic kalamata olives white skinless red snapper diced tomatoes red pepper flakes	carrots celery chicken stock garlic cloves tomatoes onion veal shanks dry red wine lemon juice	chicken broth chorizo sausage curry powder garlic greek olives ground turmeric plum tomatoes sherry wine vinegar shrimp chicken breasts artichoke hearts	carrots celery onions tomato paste veal shank dry white wine

Table 6.3: Steve's results when Ingredients and Health parameters were enabled

While comparing the results of Table 6.3 with Table 6.2, we found that enabling the *Health* parameter is beneficial for Steve because by doing this, none of the retrieved recipes violate his nutritional goals and health restrictions, and at the same time provides recipes that use the maximum number of ingredients available in Steve's kitchen. Moreover, all the recipes retrieved are of Steve's preferable cuisines (i.e., Italian and Spanish) and diet (i.e., Paleo).

Based on Ingredients, Health and Taste Preferences

Context Exploited

- Kitchen Context (Partial) i.e., name
- Personal Static and Dynamic Contexts
- Temporal Context

In this case, Steve also enabled the *Taste* parameter, which leveraged his taste preferences from his personal dynamic context to provide him with personalized recipe recommendations considering his tastes.

The recipes presented in Table 6.4 show the top five recipes (i.e., *T-R1* to *T-R5*) provided by CAPRecipes to Steve.

S.No	T-R1	T-R2	T-R3	T-R4	T-R5
Recipes	 Quick Braised Red Snapper Puttanesca	 Filetto Di Pomodoro	 Easy Lemon Pepper Scampi	 Red Snapper Puttanesca	 Broiled Shrimp Scampi
Cuisine	Italian	Italian	Italian	Italian	Italian
Diet	Paleo	Paleo	Paleo	Paleo	Paleo
Nutritional Info	Calorie:352 Protein:36g Carbs:7g Fat:13g	Calorie:481 Protein:30g Carbs:23g Fat:10g	Calorie:276 Protein:23g Carbs:19g Fat:9g	Calorie:431 Protein:36g Carbs:22g Fat:12g	Calorie:207 Protein:34g Carbs:9g Fat:8g
Used & Missing Ingredients	basil leaves capers garlic kalamata olives white skinless red snapper diced tomatoes red pepper flakes	carrots fat free chicken stock fresh basil garlic onion canned tomatoes lemon juice	garlic parsley flakes shrimp lemon pepper	bay leaf capers flat leaf parsley fresh basil garlic cloves olives onion oregano red snapper fillets tomatoes anchovies	fresh parsley garlic dry shrimp lemon juice

Table 6.4: Steve's results when Ingredients, Health and Taste parameters were enabled

After analyzing the above results, we perceived that the recipes were arranged in such an order that it first showed the recipes which matched with Steve's taste preferences, and at the same time did not violate his nutritional goals and health restrictions, and also used the maximum number of ingredients available in his kitchen.

However, when we compared the results of Table 6.4 with Table 6.3, we noted the following points:

- In Table 6.4, all the recipes retrieved i.e., *T-R1* to *T-R5* matched with Steve's personal dynamic context (i.e., taste preferences)
- In Table 6.3, *H-R2* is the only recipe which was included in the top five recipes of Table 6.4 because only *H-R2* matched with Steve's personal dynamic context.
- CAPRecipes ignored *H-R1* even though it contained zero missing ingredients and showed *T-R2* to *T-R5* which had one missing ingredient each because the logic of CAPRecipes had been formulated in such a manner that if users enable the *Taste* parameter on top of the *Health* and *Ingredient* parameters,

CAPRecipes assumes that users are more inclined to their taste preferences as opposed to ingredients used. Therefore, the system shows the recipes first which are of users' taste preferences only if they have one or two missing ingredients.

Based on Ingredients, Health, Taste Preferences and Expiry

Context Exploited

- Kitchen Context
- Personal Static and Dynamic Contexts
- Temporal Context

In this case, Steve had also enabled the *Ingredients Expiry* parameter. By doing so, all the parameters of CAPRecipes were enabled. Table 6.5 shows the top five recipes (i.e., *E-R1* to *E-R5*) retrieved for Steve.

S.No	E-R1	E-R2	E-R3	E-R4	E-R5
Recipes	 Spanish Pea Soup with Crispy Ham	 Serrano ham, olive & parsely salad	 Flavorful Chicken for Empanadas	 Zesty Italian Chicken Salad	 Spanish Lamb Chops
Cuisine	Spanish	Italian	Spanish	Italian	Spanish
Diet	Paleo	Paleo	Paleo	Paleo	Paleo
Nutritional Info	Calorie:336 Protein:16g Carbs:18g Fat:12g	Calorie:485 Protein:27g Carbs:22g Fat:13g	Calorie:347 Protein:40g Carbs:12g Fat:6g	Calorie:338 Protein:31g Carbs:16g Fat:10g	Calorie:356 Protein:39g Carbs:12g Fat:13g
Used & Missing Ingredients	chicken stock garlic serrano ham shallots peas	olives capsicum italian dressing serrano ham onion fresh parsely honey	chicken breast diced garlic cloves diced fresh parsley leaves onion empanadas	red bell pepper red onions chicken breasts cashews italian dressing asparagus spears spinach leaves	whole canned tomatoes dried basil dry white wine fresh parsley green capsicum lamb chops onion oregano mushrooms

Table 6.5: Steve's results when all parameters were enabled

Section 6.2.3 discusses the analysis and comparison of these results.

6.2.3 Evaluation

Effectiveness: *Highly*

Intended Goals when all Parameters are Enabled

When users enable all parameters, it becomes challenging for CAPRecipes as it has to provide recipes which fulfill the objectives of all parameters. Therefore, CAPRecipes first provides those recipes which contain the ingredients that are expiring soon and matches with users' taste preferences. It also takes into account that recipes do not violate users' health restrictions and nutritional goals, and uses the maximum number of available ingredients. Note that this is the ultimate goal of CAPRecipes.

For evaluating the effectiveness in terms of accuracy and personalization of results retrieved for Steve that shows the level of intended goals achieved by CAPRecipes, we compared the results of Table 6.5, where all parameters are enabled with the other results (i.e., Table 6.2 - 6.4) and noted some of the following points:

- *T-R1* was the first recommended recipe when Steve enabled the *Taste* parameter (shown in Table 6.4), but it became the sixth recommended recipe when Steve had also enabled the *Ingredients Expiry* parameter (not shown in Table 6.5) because he was presented with the option of those recipes first that contained the ingredients that were expiring soon (i.e., *E-R1* to *E-R5*).
- The recipes *E-R1* and *E-R2* were recommended first because they contained the ingredient *Serrano Ham* that had the earliest expiry date and matched with Steve's personal dynamic context.
- After *Serrano Ham*, *Chicken Breast* had the next expiry date. Therefore, CAPRecipes first showed *E-R3* and *E-R4* that contained *Chicken Breast* and than *E-R5*, which contained *Lamb* that had the next expiry date after *Chicken Breast*.
- To recommend the *Chicken Breast* recipes in Table 6.5, CAPRecipes ignored *H-R4* (shown in Table 6.3) even though it had only one missing ingredient and showed *E-R4* which had two missing ingredients because the recipe *H-R4* did not match with Steve's personal dynamic context, whereas *E-R4* did.

- CAPRecipes showed *H-R1/E-R5* in Table 6.5 as a recommendation even though *H-R1* did not match with Steve's personal dynamic context because *H-R1* had the ingredient *Lamb* that was expiring soon, did not violate his health restrictions, had no missing ingredients, and none of the recipes in his personal dynamic context included the ingredient *Lamb*.

From the above personalized recommendations as shown in Tables 6.2 - 6.5, Steve commented that he experienced high user satisfaction because with minimal effort he retrieved such a good recommendation of recipes that leveraged his numerous considerations, and he decided to cook recipe *E-R1* for dinner.

Based on the above noted points and Steve's comments, we concluded that CAPRecipes fulfilled the intended goals for Steve because he was delighted with the results, and the results retrieved were in the order that it first showed those recipes whose ingredients were expiring soon and matched with his taste preferences. Moreover, none of the recipes violated his health restrictions and nutritional goals, and the recipes provided use the maximum number of ingredients available in Steve's kitchen.

Efficiency: *0 - 1 Step (i.e., Retrieval of Personalized Recipes)*

- Without CAPRecipes, whenever Steve wanted a recipe suggestion he could cook, he had to open a web browser and apply the filters manually according to his numerous considerations (cf. Section 6.2). But, this task required a great deal of time and numerous steps. Regardless of his efforts, he was not entirely satisfied with the result.
- However, when he used CAPRecipes, he found that the process of retrieving the desired recipe recommendations by CAPRecipes is automated and took only six seconds for retrieval.
- When Steve opened CAPRecipes, by default all the parameters were enabled, and he was presented with the final output (depicted in Table 6.5) of CAPRecipes which leveraged all his contexts. The number of steps and time required to retrieve this result were zero steps and six seconds, respectively.
- The results shown in the Tables 6.2 - 6.4 are the results when Steve decided to get the recommendations based on certain parameters. It took him one step each and seven seconds to retrieve the results.

User Satisfaction: *Highly Satisfied*
To better comprehend the level of user satisfaction felt by Steve and level of objective achieved bynCAPRecipes, we had asked several quick questions after the completion of this experiment.

(a) Do you think your recommendations are enhanced by exploiting your Facebook and YouTube likes?
Answer - *Yes*

(b) Are you getting the recipes based on your preferences?
Answer - *Yes*

(c) Are the recipes considering your allergies and nutritional goals?
Answer - *Yes*

(d) Are the recipes maximizing the use of ingredients available in your kitchen?
Answer - *Yes*

(e) Are you getting the recipes according to the expiry date of ingredients?
Answer - *Yes*

(f) Do you think CAPRecipes help in promoting the reduction of food wastage?
Answer - *Yes*

(g) Will you prefer CAPRecipes over other applications?
Answer - *Yes*

(h) How satisfied are you with CAPRecipes?
Answer - *Highly Satisfied*

(i) How will you rate CAPRecipes on the scale of 1 to 10?
Answer - *9*

Based on Steve's answers to the above questions, we concluded that he was pleased with the results retrieved by CAPRecipes. He commented that he prefers to use CAPRecipes over other applications because it efficiently incorporated his considerations and will help him enhance the recipe choices he faces daily. He had also commented that it would be great if CAPRecipes could help him to do grocery shopping directly from its interface for the missing ingredients. Overall he rated CAPRecipes nine on the scale of one to ten.

6.3 Experiment 2

Neha is our second volunteer who is 25 years-old and an international graduate student at the University of Victoria. She also works part-time at Walmart, Sannich Victoria for her financial support. Despite her busy schedule, she spends at least half an hour in the gym as she is a fitness freak and wants to maintain her physique, which defines her nutritional goals. She has little time for cooking, little cooking experience and have some considerations, constraining her meal options while selecting recipes and making it troublesome for her to eat healthy. These considerations are: First, she is allergic to shrimp. Second, her nutritional goals. Third, she has certain taste preferences which she does not want to compromise. Fourth, she has pescetarian[2] diet and her preferable cuisine is Indian. Lastly, she wants to cook those recipes which maximize the use of ingredients available at her home. However, she is exhausted from manually searching for healthy recipes, and having to apply filters in light of her considerations every time. She also finds that she has to discard expired food often. Therefore, she participated to use CAPRecipes, which can leverage her context while exploiting her considerations to enhance her circumstances.

At noon of March 25, 2018, for this experiment, she decided to choose a recipe to cook for lunch with the assistance of CAPRecipes. Hence, the contexts sensed by CAPRecipes to provided her personalized recipe recommendations are illustrated below:

6.3.1 Context

Kitchen Context

It includes the ingredients available in Neha's kitchen with their expiry dates.

[2]Pescetarain is very similar to vegetarian, however it also includes the consumption of fish and seafood [Tam].

Ingredients	Expiry
coconut milk	12 days
garlic cloves	15 days
onion	17 days
tomatoes	15 days
fish sauce	30 days
ginger	12 days
bokchoy	8 days

(a)

Ingredients	Expiry
green onion	3 days
halibut fillets	6 days
red curry paste	20 days
rice	60 days
yogurt	9 days
mango	10 days
tilapia	3 days

(b)

Ingredients	Expiry
carrots	14 days
frozen peas	40 days
serrano chilli	18 days
mahimahi	2 days
red bell pepper	8 days
shallots	10 days
cod	5 days
russet potato	16 days

(c)

Table 6.6: Neha's kitchen context

Note that, for the experiment, it is assumed that herbs and seasoning used in Indian cuisines are already available in Neha's Kitchen and have a long shelf life, and are therefore not included in Table 6.6.

Personal Static Context

The following contexts depict Neha's nutritional goals, her allergy, and her preferable diet and cuisine.

- *Allergy(ies):* Shrimp
- *Nutritional Goals/day:* Calories-1800, Protein-135gm, Carbs-195gm, Fat-50gm
- *Diet:* Pescetarian
- *Cuisine(s):* Indian

The current prototype of CAPRecipes is calculating per meal nutritional goals with the following logic:

$$NutritionalGoalsPerMeal = \frac{NutritionalGoalsPerDay}{3}$$

Neha's nutritional goals per meal were: Calories-600, Protein-45gm, Carbs-65gm, Fat-16gm.

Personal Dynamic Context

It includes Neha's liked recipes' preferences extracted by exploiting her Facebook and YouTube social media profiles and includes recipes' preferences suggested by collaborative filtering.

- Facebook Preferences: *Fresh Herb and Tofu Curry*, *MahiMahi with Green Curry*, *Fish Curry with Tamarind*
- YouTube Preferences: *Thai Red Curry Fish Stew*, *Red Fish Curry*, *Grilled Salmon With Indian Spices And Raita*, *Bengali Fish Curry*, *Tilapia Baked in Green Curry*, *Coconut Curry Mussels* , *Creamy Indian-Spiced Halibut Curry*
- Similar Users Preferences: *Curry Fish Chowder With Crout-tains (paleo Croutons)*, *Mahimahi Coconut Curry Stew with Carrots and Fennel*, *Zucchini and Chickpea Green Curry*

Personal Dynamic Context

Fresh Herb and Tofu Curry, *MahiMahi with Green Curry*, *Fish Curry with Tamarind*, *Thai Red Curry Fish Stew*, *Red Fish Curry*, *Grilled Salmon With Indian Spices And Raita*, *Bengali Fish Curry*, *Tilapia Baked in Green Curry*, *Creamy Indian-Spiced Halibut Curry*, *Curry Fish Chowder With Crout-tains (paleo Croutons)*, *Mahimahi Coconut Curry Stew with Carrots and Fennel*, *Zucchini and Chickpea Green Curry*, *Coconut Curry Mussels*

Note that, Neha was quite a new user, therefore CAPRecipes had limited taste preferences in her personal dynamic context.

Temporal Context

This context includes the time, sensed by CAPRecipes, at which Neha requested recipe recommendations.

- 12:00 PM

6.3.2 Results & Analysis

This section illustrates the results retrieved for Neha when she had gradually enabled each parameter one by one. The missing ingredients of each recipe are highlighted in red, and the allergic ingredient in blue.

Based on Ingredients

Context Exploited

- Kitchen Context (Partial) i.e., name
- Personal Static Context (Partial) i.e., diet and preferable cuisines
- Temporal Context

Table 6.2 shows the top five recipes results with its necessary information, retrieved when Neha enabled only the *Ingredients* parameter.

S.No	I-R1	I-R2	I-R3	I-R4	I-R5
Recipes	 Halibut with Coconut-Red Curry	 Fish N' Chips With Curry Sauce	 Curry Fish Chowder With Crout-tains	 Fish Curry with Green Mango	 Green Curry with Bok Choy
Cuisine	Indian	Indian	Indian	Indian	Indian
Diet	Pescetarian	Pescetarian	Pescetarian	Pescetarian	Pescetarian
Nutritional Info	Calorie:250 Protein:32g Carbs:7g Fat:10g	Calorie:**800** Protein:**55g** Carbs:5g Fat:16g	Calorie:**1201** Protein:44g Carbs:63g Fat:**23g**	Calorie:410 Protein:35g Carbs:25g Fat:11g	Calorie:**658** Protein:35g Carbs:19g Fat:15g
Used & Missing Ingredients	fish sauce fresh basil fresh ginger green onions ground coriander halibut fillets light coconut milk onion red curry paste sugar	cod cornstarch curry powder fresh ginger garlic coriander seed onion red bell pepper russet potato	brown mustard seeds coconut milk cumin seeds sweet curry powder cod garam masala garlic ginger onion	cayenne pepper curry leaves ground coriander green mango onion turmeric cod	bok choy rice basil fresh ginger garlic cloves ground coriander ground cumin coconut milk serrano chiles shallots sugar cod

Table 6.7: Neha's results when only Ingredients parameter was enabled

After analyzing the results shown in Table 6.7, we recognized that the recipes *I-R1* to *I-R5* have zero missing ingredient. Thus, fulfill the intended goal of this parameter. Recipes *I-R2* and *I-R3* violates her nutritional goals as these have higher nutritional values than allowed. In addition, *Shrimp with Curry Leaves* is the eighth recommended recipe, i.e., *I-R8* (not shown in the above Table), contains her allergic

ingredient, i.e., Shrimp. Therefore, in order to get the recipes, which fulfill her nutritional goals and health restrictions, she had to enabled the *Health* parameter depicted below.

Based on Ingredients and Health

Context Exploited

- Kitchen Context (Partial) i.e., name
- Personal Static Context
- Temporal Context

The Table 6.8, shows the results of the top five recipes (i.e., *H-R1* to *H-R5*), which she was presented with by CAPRecipes when she had enabled the *Health* parameter.

S.No	H-R1	H-R2	H-R3	H-R4	H-R5
Recipes	 Halibut with Coconut-Red Curry Sauce	 Fish Curry with Green Mango	 Red Fish Curry	 Coconut Ginger Curry with Vegetables and Halibut	 Mahimahi Coconut Curry Stew with Carrots and Fennel
Cuisine	Indian	Indian	Indian	Indian	Indian
Diet	Pescetarian	Pescetarian	Pescetarian	Pescetarian	Pescetarian
Nutritional Info	Calorie:250 Protein:32g Carbs:7g Fat:10g	Calorie:410 Protein:35g Carbs:25g Fat:11g	Calorie:384 Protein:36g Carbs:21g Fat:15g	Calorie:262 Protein:12g Carbs:14g Fat:14g	Calorie:572 Protein:36g Carbs:40g Fat:16g
Used & Missing Ingredients	fish sauce fresh basil fresh ginger green onions ground coriander halibut fillets light coconut milk onion red curry paste sugar	cayenne pepper curry leaves ground coriander green mango onion turmeric cod	canned tomatoes cayenne curry leaves garlic cloves ginger ground coriander salt and pepper shallots tamarind skinless tilapia fillets turmeric	carrots cayenne coconut milk curry leaves fresh ginger ground coriander halibut onion peas serrano chiles turmeric potatoes	carrots curry leaves curry powder fresh ginger garlic cloves skinless mahimahi fillets shallots coconut milk fennel bulb

Table 6.8: Neha's results when Ingredients and Health parameters were enabled

After comparing the results of Table 6.8 with Table 6.7, we found that enabling the

Health parameter is beneficial for Neha because by doing this, none of the retrieved recipes violate her nutritional goals and health restriction, and at the same time provides recipes that use the maximum number of ingredients available in Neha's kitchen. Also, all the recipes retrieved are of Neha's preferable cuisine and diet (i.e., Indian and Pescetarian).

Based on Ingredients, Health and Taste Preferences

Context Exploited

- Kitchen Context (Partial) i.e., name
- Personal Static and Dynamic Contexts
- Temporal Context

To get the recipes based on her taste preferences, Neha then enabled the *Taste* parameter that leveraged her taste preferences from her personal dynamic context to provide her with personalized recipe recommendations.

Table 6.9 shows the top five recipes (i.e., *T-R1* to *T-R5*), presented to Neha by CAPRecipes.

S.No	T-R1	T-R2	T-R3	T-R4	T-R5
Recipes	 Red Fish Curry	 Creamy Indian-Spiced Halibut Curry	 Mahimahi with Green Curry	 Zucchini and Chickpea Green Curry	 Fish Curry with Tamarind
Cuisine	Indian	Indian	Indian	Indian	Indian
Diet	Pescetarian	Pescetarian	Pescetarian	Pescetarian	Pescetarian
Nutritional Info	Calorie:384 Protein:36g Carbs:21g Fat:15g	Calorie:558 Protein:45g Carbs:30g Fat:15g	Calorie:589 Protein:39g Carbs:50g Fat:12g	Calorie:445 Protein:13g Carbs:30g Fat:15g	Calorie:455 Protein:38g Carbs:22g Fat:16g
Used & Missing Ingredients	canned tomatoes cayenne curry leaves garlic cloves ginger ground coriander salt and pepper shallots tamarind skinless tilapia fillets turmeric	rice cayenne pepper fresh ginger garam masala garlic cloves ground coriander skinless halibut fillets onion milk yogurt turmeric heavy cream	coconut milk rice basil leaves mahimahi red bell peppers sugar fish sauce eggplant	coconut milk fish sauce sugar basil red curry paste onion garlic zucchini chickpeas	coconut milk coriander seeds cumin seeds fresh curry leaves curry powder garlic cloves onion tomatoes sugar salt tamarind skinless grouper fillets eggplant

Table 6.9: Neha's results when Ingredients, Health and Taste parameters were enabled

While analyzing the results of Table 6.9, we found that the recipes were arranged in an order that first showed the recipes, which matched with Neha's taste preferences keeping in mind that none of them violated her health restrictions and nutritional goals, and also maximized the use of ingredients available in her kitchen.

Moreover, when we compared the results of Table 6.9 with Table 6.8, we noted the following points:

- In Table 6.9, all the recipes retrieved i.e., *T-R1* to *T-R5* matched with Neha's personal dynamic context (i.e., taste preferences).
- In Table 6.8, *H-R3* is the only recipe which was included in the top five recipes of Table 6.9 because only *H-R3* matched with Neha's personal dynamic context.
- CAPRecipes ignored *H-R1, H-R2, H-R4 and H-R5* even though they contain zero or one missing ingredient and presented the recipes, i.e., *T-R2* to *T-R5*

which had one or two missing ingredients because of the logic of CAPRecipes (cf. Section 6.2.2).

Based on Ingredients, Health, Taste Preferences and Expiry

Context Exploited

- Kitchen Context
- Personal Static and Dynamic Contexts
- Temporal Context

In this case, Neha had also enabled the *Ingredients Expiry* parameter. By doing so, all the parameters of CAPRecipes were enabled. Table 6.10 shows the top five recipes (i.e., *E-R1* to *E-R5*) retrieved for Neha.

S.No	E-R1	E-R2	E-R3	E-R4	E-R5
Recipes	 Mahimahi with Green Curry	 Bengali Fish Curry (Doi Maach)	 Mahimahi Coconut Curry Stew with Carrots and Fennel	 Red Fish Curry	 Tilapia Baked in Green Curry
Cuisine	Indian	Indian	Indian	Indian	Indian
Diet	Pescetarian	Pescetarian	Pescetarian	Pescetarian	Pescetarian
Nutritional Info	Calorie:589 Protein:39g Carbs:50g Fat:12g	Calorie:232 Protein:34g Carbs:6g Fat:8g	Calorie:572 Protein:36g Carbs:40g Fat:16g	Calorie:384 Protein:36g Carbs:21g Fat:15g	Calorie:545 Protein:40g Carbs:41g Fat:16g
Used & Missing Ingredients	coconut milk rice basil leaves mahimahi red bell peppers sugar fish sauce eggplant	coriander seeds cumin seeds fresh ginger garlic clove ground turmeric mahimahi onion salt skim milk yogurt sugar shredded coconut	carrots curry leaves curry powder fresh ginger garlic cloves skinless mahimahi fillets shallots coconut milk fennel bulb	canned tomatoes cayenne curry leaves garlic cloves ginger ground coriander salt and pepper shallots tamarind skinless tilapia fillets turmeric	coconut milk fish sauce tilapia sugar green curry paste kaffir lime leaves

Table 6.10: Neha's results when all parameters were enabled

The analysis and comparison of these results are discussed in Section 6.2.3.

6.3.3 Evaluation

Effectiveness: *Moderately*

To evaluate the effectiveness, we compared results of Table 6.10 retrieved for Neha, where all the parameters were enabled with the other results (i.e., Tables 6.7 - 6.9) and noted the following points:

- *T-R1* was the first recommended recipe when Neha enabled the *Taste* parameter (shown in Table 6.9), but it became the fourth recommended recipe (i.e., *E-R4*) when she had also enabled the *Ingredients Expiry* parameter (shown in Table 6.10) because CAPRecipes presented the options of those recipes first (i.e., *E-R1* to *E-R3*) that contained the ingredient *Mahimahi* as it had the earliest expiry date.
- By leveraging Neha's kitchen context, CAPRecipes recognized that *Tilapia* had the next expiry date after *Mahimahi*. Therefore, CAPRecipes showed *E-R4* and *E-R5* as they contained the ingredient *Tilapia* and matched with Neha's personal dynamic context.
- As Neha was quite a new user of CAPRecipes, therefore her personal dynamic context was limited. When CAPRecipes leveraged her kitchen and personal dynamic contexts, it was identified that after *Tilapia*, *Cod* had the next expiry date and none of the recipes in her personal dynamic context include the ingredient *Cod*. Therefore, CAPRecipes showed some options of *Cod* recipes that were not violating her health restriction and goals and were having at the very most two missing ingredients. As the recommended *Cod* recipes were not the top five recipes, therefore not shown in Table 6.10.

From the above personalized recommendations shown in Tables 6.7 - 6.10, Neha commented that she retrieved acceptable recommendation of recipes with minimal effort as it leveraged her numerous considerations. In addition she commented that her experience was *satisfactory* and decided to cook *E-R2* for lunch.

For the result shown in Table 6.9, Neha mentioned that she was overall satisfied with this result, but she stated that the recipes *T-R4* and *T-R5* were missing the key ingredients (i.e., *Zucchini*, *Chickpeas* and *Grouper fillets*) and advised

it would be great if in the future, CAPRecipes is capable to analyze that it should not recommend such recipes which have the key ingredients missing.

Based on the above comparison and Neha's comments, we concluded that CAPRecipes moderately fulfilled the intended goals for Neha, because she was overall pleased with the results as these results were in the ascending order of expiring ingredients keeping her taste preferences into account. Also, none of the recipes violated her health restriction and nutritional goals, and used the maximum number of ingredients available in her kitchen.

Efficiency: *0 - 1 Step (i.e., Retrieval of Personalized Recipes)*

- Without CAPRecipes, whenever Neha wanted a recipe suggestion she could cook, she had to open a web browser and apply the filters manually according to her numerous considerations (cf. Section 6.3). But, this task required a great deal of time and numerous steps for the completion.
- With CAPRecipes, she learned that the process is automated to get the desired recipe recommendations and took only six seconds.
- By default all the parameters were enabled, when Neha opened CAPRecipes, and she was presented with the final output (depicted in Table 6.10) of CAPRecipes which leveraged all her contexts. The number of steps and time required to retrieve this result were zero steps and six seconds, respectively.
- Tables 6.7 - 6.9 shows the results when Neha decided to get the recommendations based on certain parameters. It took her one step each and eight seconds to retrieve the results.

User Satisfaction: *Moderately Satisfied*

To understand the level of user satisfaction felt by Neha and the level of objective achieved by CAPRecipes, we had asked several quick questions after the completion of this experiment.

(a) Do you think your recommendations are enhanced by exploiting your Facebook and YouTube likes?
Answer - *Yes*

(b) Are you getting the recipes based on your preferences?
Answer - *Yes*

(c) Are the recipes considering your allergies and nutritional goals?
Answer - *Yes*

(d) Are the recipes maximizing the use of ingredients available in your kitchen?
Answer - *Yes*

(e) Are you getting the recipes according to the expiry date of ingredients?
Answer - *Yes*

(f) Do you think CAPRecipes help in promoting the reduction of food wastage?
Answer - *Yes*

(g) Will you prefer CAPRecipes over other applications?
Answer - *Yes*

(h) How satisfied are you with CAPRecipes?
Answer - *Moderately Satisfied*

(i) How will you rate CAPRecipes on the scale of 1 to 10?
Answer -

By Neha's answers to the above questions, we inferred that she was overall pleased with the results retrieved by CAPRecipes. She mentioned that CAPRecipes had efficiently considered her considerations and will help to improve the recipe choices she faces daily. Therefore she prefers to use CAPRecipes over other applications. Overall she rated CAPRecipes eight on a scale of one to ten.

Note that, currently, CAPRecipes is evaluated based on users' data such as health information and taste preferences, where some of the data is entered by users manually. However, in future, we could use smart devices to accumulate users data automatically and to evaluate further CAPRecipes we can apply machine learning techniques when there is vast information available which can help in analyzing users' daily intake and their health for recommending personalized recipes more effectively.

6.4 Summary

This chapter evaluated CAPRecipes usability by two experiments while assessing the three quality factors: efficiency, effectiveness, and user satisfaction. In this chapter, we first presented the scenario for the two volunteers chosen for experiments, and the

context sensed by CAPRecipes for providing personalized recipe recommendations. Then, we discussed the analysis performed on the recorded results of both experiments to evaluate the level of intended goal fulfilled by CAPRecipes. Lastly, we discussed the level of user satisfaction experienced by both volunteers based on answers to the formulated quick questions.

Chapter 7

Conclusions

This chapter summarizes this thesis, outlines our contributions and presents ideas for future potential improvement to CAPRecipes.

7.1 Executive Summary

This thesis investigated the challenges endured by people every day while selecting healthy recipes, taking into account their dynamic taste preferences and the stock of ingredients in their kitchen. It also focused on how to minimize food wastage that has become the leading public concern for this world. We identified that unhealthy eating is one of the causes, raising health-related concerns such as obesity or diabetes, which account for 60% of total deaths that can be prevented by healthy eating. Lack of knowledge about how to choose healthy recipes while addressing their health restrictions and taste preferences are some of the reasons driving people to pick convenient food rather than necessary food, and this is one of the main causes of unhealthy eating.

We also identified that *health*, *taste*, *social context*, *cost* and *time* are the essential factors weighed by an individual while making food-related choices, and people manually searching for a recipe on different sites, considering their dynamic contexts and tailoring the outcomes is challenging and time-consuming.

Therefore, to deal with these various concerns, we proposed CAPRecipes as a proof of concept, which is the context-aware personalized recipe recommender system that exploits users' dynamically changing contexts to provide personalized recipe recommendations. CAPRecipes improves the user experience by automating the pro-

cess of retrieving personalized recipes, and users no longer need to manually search on the web for recipe recommendations considering their health, taste and ingredient availability.

Social media context can be utilized to provide a more personalized experience to users. Therefore, to gather users' taste preferences, CAPRecipes exploits two third-party social media applications (i.e., Facebook and YouTube), but the retrieved preferences from these applications are an eclectic list. Hence, CAPRecipes uses a natural language processing technique (i.e., text analysis) to identify the recipes' preferences from this list. We also explored a collaborative filtering algorithm which provides suggestions for recipes' taste preferences. This thesis attributed a great deal of importance to users' taste preferences because, as explained above, taste preference is one of the main causes of unhealthy eating and an essential factor in food selection. Moreover, users' taste preferences can influence their personal dietary habits and can also help reduce food wastage.

This thesis addresses the challenges of having multiple names for the same recipe or ingredient faced during recipe recommendations by finding the most relevant match using the natural language processing technique (i.e., POS tagging). CAPRecipes uses the Spoonacular API in its recommendation engine to retrieve recipes based on users' health and ingredients availability. We chose Spoonacular API as a recipe database rather than creating our own recipes database because it provides accurate results and returns a more diverse collection of recipes.

CAPRecipes provides the ability to end users to set personalization by enabling or disabling criteria at any time according to their needs, and on the basis of this personalization, recipe recommendations are updated.

We evaluated CAPRecipes qualitatively for which we conducted the experiments with two users. We recorded the results for both the experiments and performed the analysis on them, evaluating fulfilllment of our goals by CAPRecipes. To better evaluate the degree of success of CAPRecipes, we formulated several quick questions which we asked to both volunteers after the completion of their experiment. After investigating their answers, we concluded that they were pleased with their personalized results as they retrieved results with minimal manual effort, while incorporating their numerous considerations.

7.2 Contributions

This section summarizes the main contributions of this thesis.

- We developed a context-aware personalized recipe recommendation system called CAPRecipes that leverages dynamically changing user contexts (i.e., health restrictions, nutritional goals, strict dietary restrictions and moral concerns, ingredients they have in stock with their expiry, and their taste preferences) to provide personalized recipes to users.
- We explored third-party social media applications, such as YouTube and Facebook, to gather user preferences for providing personalized recipe recommendations according to user tastes.
- We explored AYLIEN Text Analysis API, which uses a natural language processing technique to help identify recipes preferences from the list of preferences retrieved from a user's social media profiles.
- To better learn the user's taste preferences, we also explored collaborative filtering techniques as it helps predict the preferences for recipes users have not liked yet.
- One limitation faced during recipe recommendations is that recipes or ingredients retrieved from different sources are lacking similar naming structure. Thus, we developed the *POS Tagging Mean Probability* algorithm that explores POS tagging, which is the natural language processing technique that maps different names of recipes or ingredients to the most relevant match.
- We tackled health-related issues and poor availability of ingredients by exploring the Spoonacular API, which is used as a recipe database in the recommendation engine of CAPRecipes.
- We developed the *Personalized Recipes Engine* algorithm, which is a hybrid-based algorithm that unifies content-based and collaborative filtering techniques along with NLP techniques for recommending personalized recipes that consider users' tastes, health and the ingredients available with their expiry information.
- We are promoting the reduction of food wastage by integrating user taste preferences and the expiry of available ingredients while recommending personalized recipes.

- We encouraged users to lead a healthy life without compromising their taste preferences as CAPRecipes considered users' allergies, nutritional goals and taste preferences in their recommendations.

- We minimized manual searching and saved a great deal of time by automating the process of retrieving personalized recipes according to numerous user considerations.

7.3 Future Work

Food is an integral part of human sustenance. However, food recommendation is the under-researched domain as compared to other domains. CAPRecipes is the proposed prototype designed to demonstrate the approach adopted to tackle concerns of users while minimizing food wastage. There is a great deal of work that can still be done to further enhance user experience and improve personalization, which helps people to live healthily while consuming the food they like. Some of them are:

- To understand user taste preferences, at present, CAPRecipes leverages two of the most popular third-party social media applications (i.e., YouTube and Facebook). CAPRecipes could be extended to leverage other social media sites such as Twitter [1] or Pinterest, [2] which helps in further learning taste preferences for providing more personalized experience to users.

- Currently, CAPRecipes supports manually entering the ingredients and expiry date when gathering kitchen context. However, CAPRecipes could be expanded to use a technique by which an ingredient and its expiry is automatically added. We have already discussed two such techniques in Section 3.2.4.

- Other than the factors explored in this research, cost and time are the two most important factors that play an essential role in the selection of recipes [FCB+96, FBS96, CBSD01]. Therefore, we could incorporate a user's daily budget and ready time in minutes (i.e., the time needed to prepare a recipe) in their personalized recipe recommendations along with their taste preferences, health, ingredients they have in the kitchen and their expiry dates.

[1] https://twitter.com
[2] https://www.pinterest.ca

- Ordinarily, people do not know the amounts of nutritional values they should consume according to their working habits. Moreover, we are in the era of Smart IoT devices such as *mobile* and *Fitbit*, which tracks users' daily routines, and based on that, could recommend nutritional values that users should consume [Kos18]. Thus, CAPRecipes could use such data to automatically add to personal static context (with consent) to improve their personalized experience.

- Currently, CAPRecipes does not track the quantities of ingredients of such recipes, which are not chosen from CAPRecipes. In such a scenario, CAPRecipes still recommends those recipes whose ingredients are already consumed. Hence, CAPRecipes could embed *FridgeCam* and *FridgeMat*, which are Smart IoT devices launched by Smarter[3] that track the quantities of ingredients left in users' refrigerators at runtime [Cri18a].

- At present, the CAPRecipes prototype is designed to recommend personalized recipes for a single user by leveraging her dynamic contexts. However, we could expand CAPRecipes for a group of users, meaning it allows users to add family members or friends to retrieve recipes that leverages the contexts of a group.

[3] https://smarter.am

Glossary

API Application Programming Interface (API), is a set of defined methods of communication for building computer software.

Eclectic likes The term eclectic likes refer to heterogeneous likes, which means it consists of likes of multiple categories.

GUI Graphical User Interface that allows users to interact with electronic devices.

HTTP HyperText Transfer Protocol (HTTP) is an application-layer protocol, which defines for collaborative, distributed, and hypermedia information systems.

JSON JavaScript Object Notation for storing and exchanging data.

NLP Natural Language Processing (NLP), is the realm of artificial intelligence that enables computer softwares to analyze and comprehend human language.

Ontology In computer science domain, Ontology is the medium to represent entities, ideas, and events, with all their interdependent properties and relations, according to a system of categories.

REST REpresentational State Transfer (REST), is an architectural style for implementing standards between computer systems on the web to make it easy for communicating with each other.

Ubiquitous computing Ubiquitous computing is a concept in software engineering and computer science where computing is made to appear anytime and everywhere.

XML eXtensible Markup Language for storing and exchanging data.

References[4]

[ADB+99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In H.-W. Gellersen, editor, *Handheld and Ubiquitous Computing (HUC 1999)*, pages 304–307, Berlin, Heidelberg, 1999. 10, 12, 13, 14

[AMW15] S. Abbar, Y. Mejova, and I. Weber. You tweet what you eat: Studying food consumption through Twitter. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI 2015)*, pages 3197–3206. ACM, 2015. 1, 2, 40

[Arn18] C. Arno. *The advantages of social media APIs*. October 2014 [Online; Accessed 29-January-2018]. URL. 17, 18, 50

[Aro18] S. Arora. *Recommendation engines: How Amazon and Netflix are winning the personalization battle | MarTech advisor*. June 2016 [Online; Accessed 12-March-2018]. URL. 60

[AT10] G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. *Recommender Systems Handbook*, page 217, 2010. 14, 15

[AWdHA+15] J. Aschemann-Witzel, I. de Hooge, P. Amani, T. Bech-Larsen, and M. Oostindjer. Consumer-related food waste: Causes and potential for action. *Sustainability*, 7(6):6457–6477, 2015. 1, 3, 40

[Ayl18a] Aylien. *Introduction · Text Analysis API | Documentation*. 2015 [Online; Accessed 31-January-2018]. URL. 18

[4] The numbers at the end of each bibliography item are links to the pages where it was cited.

[Ayl18b] Aylien. *Classification by Taxonomy · Text Analysis API | Documentation*. 2017 [Online; Accessed 31-January-2018]. URL. 19

[Ayl18c] Aylien. *Classification by Taxonomy | Text Analysis API Documentation*. February 2018 [Online; Accessed 12-March-2018]. URL. viii, 54

[Ayl18d] Aylien. *Text Analysis API | Natural Language API*. February 2018 [Online; Accessed 12-March-2018]. URL. 53

[Bak18] N. Baker. *Smart refrigerator runs apps for shopping lists, recipes*. January 2013 [Online; Accessed 01-February-2018]. URL. 23

[BBB01] M. Benerecetti, P. Bouquet, and M. Bonifacio. Distributed context-aware systems. *Human-Computer Interaction*, 16(2):213–228, 2001. 11

[BBV18] BBVAOPEN4U. *REST API: What is it, and what are its advantages in project development?* March 2016 [Online; Accessed 28-January-2018]. URL. 16

[BDR07] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007. 14

[Ben18] G. Bender. *XML vs JSON based web services: Which is the best choice?* March 2013 [Online; Accessed 29-January-2018]. URL. 17

[Bev95] N. Bevan. Human-computer interaction standards. In *Advances in Human Factors/Ergonomics*, volume 20, pages 885–890. Elsevier, 1995. 76

[BHS04] J. J. Bisgaard, M. Heise, and C. Steffensen. How is context and context-awareness defined and applied? A survey of context-awareness. *Aalborg University*, 2004. viii, 10, 11, 12, 15

[BK11] M. S. P. Babu and B. R. S. Kumar. An implementation of the user-based collaborative filtering algorithm. *International Journal of Computer Science and Information Technologies*, 2(3):1283–86, 2011. 60, 62

[Bog18] Bogdan. *Complete guide for training your own POS tagger with NLTK & Scikit-Learn*. 2013 [Online; Accessed 12-March-2018]. URL. 56

[BS97] M. Balabanović and Y. Shoham. Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997. 15, 61

[Car18] Carly Jacobs. *A household history of the fridge*. April 2013 [Online; Accessed 20-January-2018]. URL. 4

[CBSD01] M. Connors, C. A. Bisogni, J. Sobal, and C. M. Devine. Managing values in personal food systems. *Appetite*, 36(3):189–200, 2001. ix, 9, 103

[Chr18a] B. Christensen. *Paleo diet better for weight loss than nutrition recommendations | ScienceNordic*. February 2014 [Online; Accessed 12-March-2018]. URL. 77

[Chr18b] N. Chrzanowska. *Node.js vs. PHP: Which Environment To Choose For Your Next Project?* September 2017 [Online; Accessed 10-April-2018]. URL. 45

[Cri18a] R. Crist. *Smarter wants to smarten up your dumb kitchen*. 2016 [Online; Accessed 14-March-2018]. URL. 104

[Cri18b] R. Crist. *Samsung family hub refrigerator release date, price and specs - CNET*. May 2016 [Online; Accessed 25-March-2018]. URL. 33

[Cui18] L. V. Cuijk. *Why it's important that healthy food is also tasty - In love with health*. December 2015 [Online; Accessed 22-January-2018]. URL. 8

[Dee18] S. Deering. *Do you know what a REST API is?* December 2012 [Online; Accessed 28-January-2018]. URL. 16

[Dey00] A. K. Dey. Enabling the use of context in interactive applications. In *CHI'00 Extended Abstracts on Human Factors in Computing Systems (CHI EA 2000)*, pages 79–80. ACM, 2000. 10

[Dey16] A. K. Dey. *Ubiquitous Computing Fundamentals*. CRC Press, 2016. 10, 11, 12, 13, 14

[Eks18] M. Ekstrand. *Similarity functions for user-user collaborative filtering | GroupLens*. October 2013 [Online; Accessed 25-March-2018]. URL. 62

[ELL+12] K. H. Escoto, M. N. Laska, N. Larson, D. Neumark-Sztainer, and P. J. Hannan. Work hours and perceived time barriers to healthful eating among young adults. *American Journal of Health Behavior*, 36(6):786–796, 2012. 1, 2

[ES15] T. Eftimov and B. K. Seljak. Pos tagging-probability weighted method for matching the internet recipe ingredients with food composition data. In *7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015)*, volume 01, pages 330–336, Nov 2015. 41, 56

[Fac18a] *Facebook Platform - Wikipedia*. January 2018 [Online; Accessed 30-January-2018]. URL. 17, 18, 50

[Fac18b] Facebook. *Page-Graph API reference*. November 2017 [Online; Accessed 12-March-2018]. URL. 52

[FB10] J. Freyne and S. Berkovsky. Recommending food: Reasoning on recipes and ingredients. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 381–386. Springer, 2010. 3

[FBS96] L. W. Falk, C. A. Bisogni, and J. Sobal. Food choice processes of older adults: A qualitative investigation. *Journal of Nutrition Education*, 28(5):257–265, 1996. 9, 103

[FCB+96] T. Furst, M. Connors, C. A. Bisogni, J. Sobal, and L. W. Falk. Food choice: A conceptual model of the process. *Appetite*, 26(3):247–266, 1996. 9, 103

[Foo18] U. O. N. U. Food. *Food storage chart for cupboard/pantry, refrigerator and freezer | UNL food*. October 2015 [Online; Accessed 20-February-2018]. URL. 33

[GEFT+15] M. Ge, M. Elahi, I. Fernaández-Tobías, F. Ricci, and D. Massimo. Using tags and latent factors in a food recommender system. In *Proceedings of the 5th International Conference on Digital Health (DH 2015)*, pages 105–112. ACM, 2015. 20, 23

[Goo18a] Google. *Client secrets | API client library for python | Google developers*. March 2017 [Online; Accessed 12-March-2018]. URL. 47

[Goo18b] Google. *Videos: list | YouTube Data API | Google Developers*. 2018 [Online; Accessed 30-January-2018]. URL. 17

[Gri18] C. Griffith. *Samsung's smart fridge keeps track of your food stocks*. January 2018 [Online; Accessed 25-March-2018]. URL. 33

[GZR+10] I. Guy, N. Zwerdling, I. Ronen, D. Carmel, and E. Uziel. Social media recommendation based on people and tags. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2010)*, pages 194–201. ACM, 2010. 1, 2, 40

[Har18] Harrison. *Tokenizing words and sentences with NLTK*. 2017 [Online; Accessed 12-March-2018]. URL. 57

[Hew18] N. Hewitt. *15 Delicious and healthy recipes for lazy people*. 2017 [Online; Accessed 20-January-2018]. URL. 8

[Hof14] D. Hofstrand. Can we meet the world's growing demand for food. *AgMRC Renewable Energy & Climate Change Newsletter*, 2014. 4

[Inv18] Investopedia. *Natural Language Processing (NLP)*. 2017 [Online; Accessed 30-January-2018]. URL. 18

[JL18] S. O. Jaromir Latal, Raja Upadhyay. *LeChefu*. November 2017 [Online; Accessed 12-March-2018]. URL. 44, 66

[Kee18] T. Keenan. *What is collaborative filtering? - Hiring | Upwork*. December 2017 [Online; Accessed 25-March-2018]. URL. 61

[Kos18] D. Kosecki. *Ask Fitbit: Why does my calorie allowance change throughout the day?* 2018 [Online; Accessed 28-March-2018]. URL. 104

[Lac18] Lacey Baier. *How to eat healthy and not hate your food.* January 2016 [Online; Accessed 20-January-2018]. URL. 8

[Lau18] L. S. Laughlin. *Netflix named one of the world's most admired companies | Fortune.* March 2016 [Online; Accessed 25-March-2018]. URL. 70

[LKAA96] S. Long, R. Kooper, G. D. Abowd, and C. G. Atkeson. Rapid prototyping of mobile context-aware applications: The cyberguide case study. In *Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking (MobiCom 1996)*, pages 97–107. ACM, 1996. 15

[LSH+97] R. Lappalainen, A. Saba, L. Holm, H. Mykkanen, M. Gibney, and A. Moles. Difficulties in trying to eat healthier: Descriptive analysis of perceived barriers for healthy eating. *European Journal of Clinical Nutrition*, 51(2):S36, 1997. 1, 2

[LZW13] H. Li, S. Zhang, and X. Wang. A personalization recommendation algorithm for e-commerce. *Journal Of Software*, 8(1):176–183, 2013. 60

[MAD00] D. R. Morse, S. Armstrong, and A. K. Dey. The what, who, where, when, why and how of context-awareness. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems*, CHI EA 2000, pages 371–371, New York, NY, USA, 2000. ACM. 10

[Mar18] S. Marafi. *Collaborative filtering with python: Salem Marafi.* April 2015 [Online; Accessed 25-March-2018]. URL. ix, 61

[Mas18] Mashape. *Recipe - Food - Nutrition API Documentation.* August 2015 [Online; Accessed 21-January-2018]. URL. viii, 19, 68

[Nat18] *Natural Language Toolkit.* 2017 [Online; Accessed 12-March-2018]. URL. 57

[NPS03] E. Newcomb, T. Pashley, and J. Stasko. Mobile computing in the retail arena. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2003)*, pages 337–344. ACM, 2003. 15

[OBB+90] D. Ornish, S. E. Brown, J. Billings, L. Scherwitz, W. T. Armstrong, T. A. Ports, S. M. McLanahan, R. L. Kirkeeide, K. Gould, and R. Brand. Can lifestyle changes reverse coronary heart disease?: The lifestyle heart trial. *The Lancet*, 336(8708):129–133, 1990. 1

[oC18] G. of Canada. *Safe food storage - Canada.ca*. November 2014 [Online; Accessed 20-February-2018]. URL. 33

[Oli18] Olio. *The problem of food waste*. January 2018 [Online; Accessed 20-January-2018]. URL. 3

[oPH18] H. T. C. S. of Public Health. *Healthy Eating Plate & Healthy Eating Pyramid | The Nutrition Source*. September 2011 [Online; Accessed 23-January-2018]. URL. ix, 9, 10

[Per18] S. Perez. *Consumers Spend 85% Of Time On Smartphones In Apps, But Only 5 Apps See Heavy Use | TechCrunch*. June 2015 [Online; Accessed 10-April-2018]. URL. 46

[Pie18] D. Pierce. *We're drowning in content. Recommendations are what we need | WIRED*. April 2016 [Online; Accessed 31-January-2018]. URL. 17

[PRM00] J. Pascoe, N. Ryan, and D. Morse. Using while moving: Hci issues in fieldwork environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(3):417–437, 2000. 10

[Pro18] ProgrammableWeb. *Food API*. 2017 [Online; Accessed 31-January-2018]. URL. 19

[Pub18] H. H. Publishing. *Questions and answers about the Healthy Eating Plate - Harvard Health*. September 2011 [Online; Accessed 23-January-2018]. URL. 9

[Rap18a] RapidAPI. *FAQs - Mashape Marketplace*. 2017 [Online; Accessed 31-January-2018]. URL. 20

[Rap18b] RapidAPI. *RapidAPI Marketplace | Recipe - Food - Nutrition*. 2018 [Online; Accessed 31-January-2018]. URL. 19

[Rec18] *Recommender system - Wikipedia*. February 2018 [Online; Accessed 25-March-2018]. URL. 70

[RRS11] F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer, 2011. 14

[RWM14] A. Rung, F. Warnke, and N. Mattheos. Investigating the use of smartphones for learning purposes by australian dental students. *JMIR mHealth and uHealth*, 2(2), 2014. 46

[SAK+91] B. Senauer, E. Asp, J. Kinsey, et al. *Food trends and the changing consumer*. Eagan Press, 1991. 3

[SAM18] SAMSUNG - NWESROOM. *Samsung introduces an entirely new category in refrigeration as part of kitchen appliance lineup at CES 2016*. January 2016 [Online; Accessed 20-January-2018]. URL. 4

[Sar18] Sarah Perez. *Vevo's recommendations get more personalized, thanks to integrations with Spotify, Twitter and YouTube | TechCrunch*. March 2016 [Online; Accessed 20-January-2018]. URL. 2

[SAW94] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *First Workshop on Mobile Computing Systems and Applications (WMCSA 1994)*, pages 85–90. IEEE, 1994. 10, 11

[SBL] E. K. Steven Bird and E. Loper. *Categorizing and tagging words*. Online; Accessed 12-March-2018. 56

[SBT+10] N. Suksom, M. Buranarach, Y. M. Thein, T. Supnithi, and P. Netisopakul. A knowledge-based framework for development of personalized food recommender system. In *Proceedings of the 5th International Conference on Knowledge, Information and Creativity Support Systems*, 2010. 23

[SGT10] B. Scheibehenne, R. Greifeneder, and P. M. Todd. Can there ever be too many options? A meta-analytic review of choice overload. *Journal of Consumer Research*, 37(3):409–425, 2010. 3

[Sid] A. Siddharthan. *Recommender systems 2: Collaborative filtering*. 60

[Spe18] M. Specter. *Can fast food get healthy?* November 2015 [Online; Accessed 21-January-2018]. URL. 8

[Spo18] Spoonacular. *Food and Recipe API*. 2017 [Online; Accessed 31-January-2018]. URL. 19, 66

[Ste18] F. V. Steeg. *Context-aware recommender systems*. 2015 [Online; Accessed 26-January-2018]. URL.

[Tam] Tamryn. *Pescetarian vs Vegetarian: Which diet is healthier for you?* 88

[TB14] H. Z. Tang and T. Bresnahan. The collaborative filtering effect of netflix ratings for Indie films versus blockbusters and heavy users versus casual users. 2014. 60, 61

[TD09] H.-L. Truong and S. Dustdar. A survey on context-aware web service systems. *International Journal of Web Information Systems*, 5(1):5–31, 2009. 14

[TE17] C. Trattner and D. Elsweiler. Food recommender systems: Important contributions, challenges and future research directions. *arXiv preprint arXiv:1711.02760*, 2017. viii, 1, 3, 20, 21, 22, 41, 44

[Tec18a] S. Technologies. *Natural Language Processing (NLP) services*. 2017 [Online; Accessed 31-January-2018]. URL. 18

[Tec18b] Techopedia. *What is collaborative filtering (CF)? - Definition from Techopedia*. January 2017 [Online; Accessed 25-March-2018]. URL. 60, 61

[TUMN16] N. Takata, M. Ueda, Y. Morishita, and S. Nakajima. Automatic recipe metadata generating method by considering users' various moods. In *Proceedings of the International Multiconference of Engineers and Computer Scientists 2016*, pages 413–418, 2016. 23

[Tut18a] R. A. Tutorial. *What is REST*. 2015 [Online; Accessed 28-January-2018]. URL. 16

[Tut18b] R. A. Tutorial. *HTTP Methods for RESTful services*. 2017 [Online; Accessed 29-January-2018]. URL. 16

[Use18] *User interface - Wikipedia*. February 2013 [Online; Accessed 12-March-2018]. URL. 38

[UTN11] M. Ueda, M. Takahata, and S. Nakajima. User's food preference extraction for personalized cooking recipe recommendation. In *Workshop of International Semantic Web Conference*, pages 98–105, 2011. 3, 23

[Vaq18] M. Vaqqas. *RESTful web services: A tutorial | Dr Dobb's*. September 2014 [Online; Accessed 28-January-2018]. URL. 16

[VDLG10] K. Verbert, E. Duval, S. Lindstaedt, and D. Gillet. Context-aware recommender systems. *Journal of Universal Computer Science*, 16(16):2175–2178, 2010. 14, 15

[VMO+12] K. Verbert, N. Manouselis, X. Ochoa, M. Wolpers, H. Drachsler, I. Bosnic, and E. Duval. Context-aware recommender systems for learning: A survey and future challenges. *IEEE Transactions on Learning Technologies*, 5(4):318–335, 2012. 14

[WDVR06] J. Wang, A. P. De Vries, and M. J. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2006)*, pages 501–508. ACM, 2006. 62

[Web] WebSystique. *Spring MVC 4 RESTFul web services CRUD Example+RestTemplate - WebSystique*. Online; Accessed 28-January-2018. 16, 17

[Wor18a] World Food Programme. *Zero Hunger*. January 2018 [Online; Accessed 20-January-2018]. URL. 3

[Wor18b] World Health Organization. *WHO | Obesity and overweight*. February 2018 [Online; Accessed 20-January-2018]. URL. 2

[WWO+12] H. Williams, F. Wikström, T. Otterbring, M. Löfgren, and A. Gustafsson. Reasons for household food waste with special attention to packaging. *Journal of Cleaner Production*, 24:141–148, 2012. 4

[X+16] Y. Xiao et al. Web front-end architecture with Node js. platform. 2016. 45

[You18a] *YouTube API - Wikipedia*. November 2017 [Online; Accessed 30-January-2018]. URL. 17

[You18b] *YouTube - Wikipedia*. March 2018 [Online; Accessed 12-March-2018]. URL.

[You18c] YouTube. *Videos: list | YouTube Data API | Google Developers*. November 2017 [Online; Accessed 12-March-2018]. URL. viii, 48, 49

Appendix A

Source Code

Source Code A.1: Python code to retrieve users' YouTube likes and to identify the category of each YouTube like. Explanation of this code is in Section 4.1.1 and 4.2.1.

```
# Implementation as described in YouTube Developer API guide
# -*- coding: utf-8 -*-
import os
import json
import google.oauth2.credentials
import pprint
import google_auth_oauthlib.flow
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
from google_auth_oauthlib.flow import InstalledAppFlow
from aylienapiclient import textapi

# The CLIENT_SECRETS_FILE variable specifies the name of a file that contains
# the OAuth 2.0 information for this application, including its client_id and
# client_secret.
CLIENT_SECRETS_FILE = "client_secret.json"

# This OAuth 2.0 access scope allows for full read/write access to the
# authenticated user's account and requires requests to use an SSL connection.
SCOPES = ['https://www.googleapis.com/auth/youtube.force-ssl']
API_SERVICE_NAME = 'youtube'
API_VERSION = 'v3'

def get_authenticated_service():
  flow = InstalledAppFlow.from_client_secrets_file(CLIENT_SECRETS_FILE, SCOPES)
  credentials = flow.run_console()
  return build(API_SERVICE_NAME, API_VERSION, credentials = credentials)

def print_response(response):
  print(response)

# Build a resource based on a list of properties given as key-value pairs.
```

```
# Leave properties with empty values out of the inserted resource.
def build_resource(properties):
  resource = {}
  for p in properties:
    # Given a key like "snippet.title", split into "snippet" and "title", where
    # "snippet" will be an object and "title" will be a property in that object.
    prop_array = p.split('.')
    ref = resource
    for pa in range(0, len(prop_array)):
      is_array = False
      key = prop_array[pa]

      # For properties that have array values, convert a name like
      # "snippet.tags[]" to snippet.tags, and set a flag to handle
      # the value as an array.
      if key[-2:] == '[]':
        key = key[0:len(key)-2:]
        is_array = True

      if pa == (len(prop_array) - 1):
        # Leave properties without values out of inserted resource.
        if properties[p]:
          if is_array:
            ref[key] = properties[p].split(',')
          else:
            ref[key] = properties[p]
      elif key not in ref:
        # For example, the property is "snippet.title", but the resource does
        # not yet have a "snippet" object. Create the snippet object here.
        # Setting "ref = ref[key]" means that in the next time through the
        # "for pa in range ..." loop, we will be setting a property in the
        # resource's "snippet" object.
        ref[key] = {}
        ref = ref[key]
      else:
        # For example, the property is "snippet.description", and the resource
        # already has a "snippet" object.
        ref = ref[key]
  return resource

# Remove keyword arguments that are not set
def remove_empty_kwargs(**kwargs):
  good_kwargs = {}
  if kwargs is not None:
    for key, value in kwargs.iteritems():
      if value:
        good_kwargs[key] = value
  return good_kwargs

# Function to retrieve likes from Users' YouTube Profile
def videos_list_my_rated_videos(client, **kwargs):
  kwargs = remove_empty_kwargs(**kwargs)

```

```
  response = client.videos().list(
    **kwargs
  ).execute()

  return response

# To identify recipe likes retrieved from the eclectic list of YouTube likes using Aylien Text Analysis API
def aylienTextAnalysis(jsonData):
  client = textapi.Client("API_ID", "API_KEY")
  recipeVideos = []
  for eachVideo in jsonData:
  # passing description attribute of each YouTube like as text query parameter and iab-qag taxonomy attribute as a path parameter
  classifications = client.ClassifyByTaxonomy({"text": eachVideo["description"].encode('utf-8'), "taxonomy": "iab-qag"})
  for category in classifications['categories']:
    if category["label"] == "Food & Drink":
      # Add as a recipe like if description belongs to Food and Drink category
      title = eachVideo["title"].encode('utf-8')
      recipeAttr = title.split("|")
      recipeVideos.append(recipeAttr[0].rstrip())

  return recipeVideos

if __name__ == '__main__':
  # When running locally, disable OAuthlib's HTTPs verification. When
  # running in production *do not* leave this option enabled.
  os.environ['OAUTHLIB_INSECURE_TRANSPORT'] = '1'
  client = get_authenticated_service()
  # calling function to retrieve users' likes by passing query parameters
  jsonResponse = videos_list_my_rated_videos(client, part='snippet', myRating='like', maxResults=50)
  # filtering the result to keep only description, title and id of each liked video.
  jsonVideos = []
  for eachVideo in jsonResponse["items"]:
  eachjsonVideo = {}
  eachjsonVideo["id"] = eachVideo["id"]
  for key,value in eachVideo["snippet"].items():
    if(key =="title"):
      eachjsonVideo["title"] = value
    elif(key == "tags"):
      eachjsonVideo["tags"] = value
    elif(key == "description"):
      eachjsonVideo["description"] = value
  jsonVideos.append(eachjsonVideo)
  # calling Aylien text analysis function to identify likes
  recipeJsonVideos = aylienTextAnalysis(jsonVideos)

  youtubeLikes = {}
  youtubeLikes["youtubeRecipeLikes"] = recipeJsonVideos
  with open('youtubeRecipeLikes.json', 'w') as outfile:
    json.dump(youtubeLikes, outfile)
```

```
  with open('youtubeEclecticLikes.json', 'w') as outfile:
    json.dump(jsonVideos, outfile)
```

Source Code A.2: Python code to retrieve users' Facebook likes and to identify the category of each facebook like. Explanation of this code is in Section 4.1.2 and 4.2.1.

```
# Implementation of extracting recipes' likes using Facebook Graph API and Aylien Text Analysis API
import os
import json
import urllib
import pprint
from aylienapiclient import textapi

global ACCESS_TOKEN

# To identify recipes' likes from eclectic list of Facebook's likes using Aylien Text Analysis API
def aylienTextAnalysis(jsonData):
  client = textapi.Client("API_ID", "API_KEY")
  recipeVideos = []
  for eachLike in jsonData["data"]:
  # retrieving additional information for each like using Facebook Graph API
  host = "https://graph.facebook.com"
  path = "/"+eachLike["id"].rstrip()

  params = urllib.urlencode({"access_token": ACCESS_TOKEN,"fields":"about,description"})
  url = "{host}{path}?{params}".format(host=host, path=path, params=params)
  resp = urllib.urlopen(url).read()
  with open('FacebookEachLike.json', 'w') as outfile:
    json.dump(resp, outfile)
  pageDes =  json.loads(resp)
  for key, value in pageDes.iteritems():
    classifications ={}
    # identifying if like contain description or about
    if key == "description" or key == "about":
      # passing description attribute of each Facebook like as text query parameter and iab-qag taxonomy attribute as a path parameter
      classifications = client.ClassifyByTaxonomy({"text": value.encode('utf-8'), "taxonomy": "iab-qag"})
      break
  if len(classifications) >=1:
    for category in classifications['categories']:
      if category["label"] == "Food & Drink":
        # Add as a recipe like if the description or about belongs to Food and Drink category
        recipeVideos.append(eachLike["name"])
  return recipeVideos

# build the URL for the API endpoint
host = "https://graph.facebook.com"
```

```
path = "/{user-id}/likes"
ACCESS_TOKEN = "ACCESS_TOKEN"
params = urllib.urlencode({"access_token": ACCESS_TOKEN})

url = "{host}{path}?{params}".format(host=host, path=path, params=params)

# open the URL and read the response
resp = urllib.urlopen(url).read()

facebookEclecLikes = json.loads(resp)
# calling aylienTextAnalysis function to identify recipes' likes
facebookRecipeLikes = aylienTextAnalysis(facebookEclecLikes)
# writing response data to files
with open('facebookEclecticLikes.json', 'w') as outfile:
    json.dump(facebookEclecLikes, outfile)
facebookLikes = {}
facebookLikes["facebookRecipeLikes"] = facebookRecipeLikes
with open('facebookRecipeLikes.json', 'w') as outfile:
    json.dump(facebookLikes, outfile)
```

Source Code A.3: Python code of POS Tagging Mean Probability algorithm to find the most relevant match for recipes or ingredients. Explanation of this code is in Section 4.2.2.

```
from nltk import pos_tag, word_tokenize
import sys
import json
import urllib2
import requests
import xml.etree.ElementTree as ET
from xml.etree.ElementTree import XML, fromstring, tostring
import difflib
import ast
import re

# To apply POS tagging and extract nouns, verbs and adjectives
def posTagResponse(recipe):
  posResult = pos_tag(word_tokenize(recipe))
  posTagJSON = {}
  noun = []
  verb = []
  adjective = []
  for tagTuple in posResult:
    if(tagTuple[1] == "NN" or tagTuple[1] == "NNS" or tagTuple[1] == "NNP" or
    tagTuple[1] == "NNPS" ):
      noun.append(tagTuple[0].lower())
    elif(tagTuple[1] == "VB" or tagTuple[1] == "VBD" or tagTuple[1] == "VBG" or
    tagTuple[1] == "VBN"  or tagTuple[1] == "VBP"  or tagTuple[1] == "VBZ" ):
      verb.append(tagTuple[0].lower())
    elif(tagTuple[1] == "JJ" or tagTuple[1] == "JJR" or tagTuple[1] == "JJS" ):
      adjective.append(tagTuple[0].lower())
  if(len(noun)>=1):
```

```
27      posTagJSON["Noun"] = noun
28    if(len(verb)>=1):
29      posTagJSON["Verb"] = verb
30    if(len(adjective)>=1):
31      posTagJSON["Adjective"] = adjective
32
33    return posTagJSON
34
35  # To calculate the probability between nouns, verbs and adjectives of both the list
      of recipes
36  def probabilityCalculation(tag, firstRecipesJSON, secondRecipesJSON):
37    counter = 0
38    sum = 0.0
39    prob  = 1.0
40    if(tag in firstRecipesJSON and tag in secondRecipesJSON):
41      for tagR1 in firstRecipesJSON[tag]:
42        for tagR2 in secondRecipesJSON[tag]:
43          p = difflib.SequenceMatcher(None,tagR1,tagR2).ratio()
44          if(p>0.7):
45            sum += p
46          counter+= 1
47      if(counter>0):
48        prob = sum /counter
49
50    return prob
51
52  if __name__ == "__main__":
53    recipeName = sys.argv[1]
54    # calling function to get set of nouns, verbs amd adjectives
55    recipeNameJSON  = posTagResponse(recipeName)
56    response_body = eval(sys.argv[2])
57    sum  = 0.0
58    counter = 0
59    pairs = {}
60    # setting max mean probability to 0.7
61    maxProbability  = 0.7
62    nounProbability = 0
63    verbProbability = 0
64    adjProbability = 0
65    similarRecipe = ""
66    for eachRecipe in response_body:
67      # calling function to get set of nouns, verbs amd adjectives
68      eachrecipeJSON = posTagResponse(eachRecipe)
69      # calculating the noun, verb and adjectives probability
70      nounProbability = probabilityCalculation("Noun", recipeNameJSON, eachrecipeJSON)
71      verbProbability = probabilityCalculation("Verb", recipeNameJSON, eachrecipeJSON)
72      adjProbability = probabilityCalculation("Adjective", recipeNameJSON,
        eachrecipeJSON)
73
74      totalProbability =  (nounProbability + verbProbability+ adjProbability)/3
75
76      if(totalProbability> maxProbability):
77        maxProbability = totalProbability
```

```
78        similarRecipe = eachRecipe
79
80    print similarRecipe
```

Source Code A.4: Python code to integrate preferences retrieved from users' social media profiles and collaborative filtering into the users' personal dynamic context. Explanation of this code is in Section 4.4.

```
1  import sys
2  import json
3  import subprocess
4
5  # Function to find the most relevant match using POS tagging
6  def findMostRelevantMatch(recipe, personalContextData):
7    # calling POS tagging script to find the match
8    match =   subprocess.check_output([sys.executable,"./posTaggingRecipeMatch.py",
9    recipe, str(personalContextData)])
10   match = match.rstrip()
11   if(len(match)<1):
12     # appending the matched recipes
13     personalContextData.append(recipe)
14   return personalContextData
15
16 # To update the personal dynamic context based on social media and other similar
       users likes
17 if __name__ == "__main__":
18   # loading youtube recipe likes
19   YTRecipeLikesJSON = json.load(open('./youtubeRecipeLikes.json'))
20   YTRecipeLikes = YTRecipeLikesJSON["youtubeRecipeLikes"]
21   # loading personal dynamic context
22   personalContextJSON = json.load(open('./Personal_context.json'))
23   # loading facebook recipe likes
24   FBRecipeLikesJSON = json.load(open('./facebookRecipeLikes.json'))
25   FBRecipeLikes = FBRecipeLikesJSON["facebookRecipeLikes"]
26   # loading suggested likes based on similar users
27   collRecipeLikesJSON = json.load(open('./collaborativeRecipesLikes.json'))
28   collRecipeLikes = collRecipeLikesJSON["colloborativeRecipeLikes"]
29   personalContextRecipeLikes = []
30   recipeLikes = YTRecipeLikes + FBRecipeLikes + collRecipeLikes
31
32   for key,val in personalContextJSON.iteritems():
33     if key == "Dynamic Context":
34       personalContextRecipeLikes = val["Recipe Preferences"]
35
36   for eachRecipe in recipeLikes:
37     if(len(personalContextRecipeLikes)>=1):
38       # calling function to find the most revelant match to eliminate duplicates
       before integrating
39       personalContextRecipeLikes = findMostRelevantMatch(eachRecipe,
       personalContextRecipeLikes)
40
41   # Integrating recipe likes of social media and similar user in users' personal
```

```
    dynamic context
  personalRecipeJSON ={}
  personalRecipeJSON["Recipe Preferences"] = list(set(personalContextRecipeLikes))
  personalContextJSON["Dynamic Context"] = personalRecipeJSON
  with open('./Personal_context.json', 'w') as outfile:
    json.dump(personalContextJSON, outfile)
```

Source Code A.5: Python code of Personalized Recipes Engine algorithm that provides the final personalized recipes list. Explanation of this is in Section 5.2.

```
# !/usr/bin/python27

# Usage for getting recipelist: http://localhost:3020/CAPRecipes/recipelist?type=&
    cuisine=&userHealthContext=True&userKitchenContext=True&userTasteContext=True&
    userAvailableIngredientExpiry=True
# Usage for getting cookingstep: http://localhost:3020/CAPRecipes/cookingsteps?id
    =547899

import psycopg2
import pprint
import unirest
import io
import json
import re
from bottle import Bottle, route, run, request, response
from Queue import Queue
from threading import Thread
import sys
from difflib import SequenceMatcher
from datetime import datetime
import subprocess
import datetime as DT
import operator

app = Bottle()

@app.hook('after_request')
def enable_cors():
  """
  You need to add some headers to each request.
  Don't use the wildcard '*' for Access-Control-Allow-Origin in production.
  """
  response.headers['Access-Control-Allow-Origin'] = '*'
  response.headers['Access-Control-Allow-Methods'] = 'PUT, GET, POST, DELETE, OPTIONS
    '
  response.headers['Access-Control-Allow-Headers'] = 'Origin, Accept, Content-Type, X
    -Requested-With, X-CSRF-Token'

@app.route('/CAPRecipes/<name>', method=['GET'])
def recdata(name):
  if name == "recipelist":
    # reading dishtype as query parameter
```

```
38      dishtype = request.query.type
39      # if dishtype not selected then reads from users' temporal context
40      if(len(dishtype)<=1):
41        # calling function to get dish type according to users' temporal context (i.e.,
         current meal time)
42        dishtype = getDishType()
43      recipeCount = "40"
44      # reading query parameters from url
45      cuisineinfo = request.query.cuisine
46      userTastePref =  request.query.userTasteContext
47      userHealthPref = request.query.userHealthContext
48      userKitchenContextPref = request.query.userKitchenContext
49      userIngrExpiryPref =  request.query.userAvailableIngredientExpiry
50      maxProteins = ""
51      maxCarbs = ""
52      maxFat = ""
53      maxCalories = ""
54      allergies = ""
55      ingredientsString= {}
56
57      # reading users' personal static and dynamic context variables
58      userPersonalContextJSON = json.load(open('./harshit-python/Personal_context.json'
        ))
59      # if health toggle button is enabled
60      if(bool(userHealthPref) == True):
61        allergies = userPersonalContextJSON["Static Context"]["Allergies"]
62        maxProteins = str((userPersonalContextJSON["Static Context"]["Nutritional Goals
        "]["Proteins"])/3)
63        maxCarbs = str((userPersonalContextJSON["Static Context"]["Nutritional Goals"][
        "Carbs"])/3)
64        maxFat = str((userPersonalContextJSON["Static Context"]["Nutritional Goals"]["
        Fat"])/3)
65        maxCalories = str((userPersonalContextJSON["Static Context"]["Nutritional Goals
        "]["Calories"])/3)
66      diet = userPersonalContextJSON["Static Context"]["Diet"]
67      # if cuisine not selected then reads from users' personal static context
68      if(len(cuisineinfo)<=1):
69        cuisineinfo = userPersonalContextJSON["Static Context"]["Cuisine"]
70
71      # if Ingredients toggle button is enabled
72      if(bool(userKitchenContextPref) == True):
73        ingredientsJSON = {}
74        # reading users' kitchen context (i.e., available ingredients)
75        ingredientsJSON = json.load(open('./harshit-python/kitchen_context.json'))
76        # calling the script to sort the ingredients
77        ingredientsSortedJSON = subprocess.check_output([sys.executable,"./harshit-
        python/sort_date.py",str(ingredientsJSON)])
78        ingredientsArr =[]
79        for ref in ingredientsJSON["Kitchen Context"]:
80          for ingr in ref["Ingredients"]:
81            ingredientsArr.append(ingr)
82
83        ingredientsString = '%2C+'.join(ingredientsArr)
```

```
84      ingredientsString = ingredientsString.replace(" ","%20")
85
86    dishtype.replace ("%20", "+")
87    cuisineinfo.replace ("%2C", ",")
88
89    # calling Spoonacular API to retrieve recipes
90    apicall = "https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/
      searchComplex?addRecipeInformation=true&cuisine="+cuisineinfo+"&diet="+diet+"&
      excludeIngredients=&fillIngredients=true&includeIngredients="+ingredientsString+"
      &intolerances="+allergies+"&limitLicense=false&maxCalories="+maxCalories+"&
      maxCarbs="+maxCarbs+"&maxFat="+maxFat+"&maxProtein="+maxProteins+"&number="+
      recipeCount+"&offset=0&ranking=2&type="+dishtype
91
92    response = unirest.get(apicall,
93    headers={
94    "X-Mashape-Key": "Developer_API_Key",
95    "Accept": "application/json"
96    }
97    )
98    with io.open('recjsonone.json', 'w', encoding='utf-8') as f:
99      f.write(unicode(json.dumps(response.body,ensure_ascii=False)))
100
101   with open('recjsonone.json') as data_file:
102     data = json.load(data_file)
103
104   for val in data:
105     if val == 'results':
106       listrecm = []
107       for recipe in data[val]:
108         if len(recipe) <=13:
109           continue
110         if len(recipe['analyzedInstructions']) > 0:
111           recipelistgen(recipe, listrecm, recipe['pricePerServing'])
112   with io.open('recresponse.json', 'w', encoding='utf-8') as f:
113     f.write(unicode(json.dumps(listrecm,ensure_ascii=False)))
114   for item in listrecm:
115     del item['cookingInstructions']
116
117   # if Taste toggle button is enabled
118   if(bool(userTastePref) == True):
119     # calling function to refines and sorts recipes based on users' taste
      preferences
120     listrecm = recipesRefiningWithTaste(listrecm, userPersonalContextJSON)
121
122   # if Ingredients Expiry toggle button is enabled
123   if(bool(userIngrExpiryPref) == True):
124     # calling function to refines and sorts recipes based on Ingredients Expiry
125     listrecm = recipesRefiningWithExpiry(listrecm, ingredientsSortedJSON)
126   return dict(data=listrecm)
127
128 # To generate recipe instruction page
129  elif name == "cookingsteps":
130   with open('recresponse.json') as data_file:
```

```
131       listrecm = json.load(data_file)
132     recipeid = request.query.id
133     listrecm1 = []
134 
135     for item in listrecm:
136       if str(item['id']).strip() != str(recipeid).strip():
137         listrecm1.append(listrecm.index(item))
138 
139     for v in listrecm1[::-1]:
140       del listrecm[v]
141 
142     return dict(data=listrecm)
143 
144 # Function to get sorted recipes list based on users' taste
145 def recipesRefiningWithTaste(recList, personalDynContextJSON):
146   usersRecipePreferences = personalDynContextJSON["Dynamic Context"]["Recipe
      Preferences"]
147   sortedRecipeList = []
148   notMatchedRecList = []
149   for eachRecipe in recList:
150     # calling the POS tagging script to find the most relevant match of recipes
151     match =   subprocess.check_output([sys.executable,"./harshit-python/
      posTaggingRecipeMatch.py",
152   eachRecipe["title"], str(usersRecipePreferences)])
153     match = match.rstrip()
154     if(len(match)>5):
155       sortedRecipeList.append(eachRecipe)
156     else:
157       notMatchedRecList.append(eachRecipe)
158   return sortedRecipeList + notMatchedRecList
159 
160 # Function to refine recipes list based on ingredients expiry
161 def recipesRefiningWithExpiry(recList, expiringIngredients):
162   finalRecList = []
163   for eachRecipe in recList:
164     cntr = 0
165     date = ''
166     for usedattr in eachRecipe['usedIngredients']:
167       # calling the POS tagging script for matching the ingredients of users' kitchen
       context with Spoonacular ingredients
168       dateIngr = subprocess.check_output([sys.executable,"./harshit-python/
      posTaggingIngrdMatch.py",
169       usedattr['name'], expiringIngredients])
170       if(len(dateIngr) > 2):
171         cntr += 1
172         if(date == '' or date > dateIngr):
173           date = dateIngr
174     if(cntr == 0):
175       today = DT.date.today()
176       week_after = today + DT.timedelta(days=11)
177       date = datetime.strptime(str(week_after), '%Y-%m-%d').strftime('%m/%d/%y')
178     eachRecipe['date'] = date
179     eachRecipe['expiringIngredients'] = cntr
```

```
180     finalRecList.append(eachRecipe)
181   #ORDER BY date ASC, expiringIngredients DESC (using SQL notation) for a list of
        dictionaries can be done like this:
182   finalRecList.sort(key = operator.itemgetter('expiringIngredients'), reverse=True)
183   finalRecList.sort(key = operator.itemgetter('date'))
184   return finalRecList
185
186 # Function to get dish type according to users' temporal context (i.e., current meal
      time)
187 def getDishType():
188   currentDishType = ""
189   currentMealTime = int(datetime.now().strftime('%H'))
190   if(currentMealTime>=3 and currentMealTime<=10):
191     currentDishType = "breakfast"
192   elif(currentMealTime>=11 and currentMealTime<=16):
193     currentDishType = "lunch"
194   else:
195     currentDishType = "main%20course"
196   return "breakfast"
197 # Function that generates final personalized recipes list
198 def recipelistgen(origrecipe,reclist,prcperserving):
199   rec = {}
200   rec['id'] = origrecipe['id']
201   rec['costPerServing'] = round(prcperserving/100,2)
202   rec['title'] = origrecipe['title']
203   rec['missedIngredients'] = origrecipe['missedIngredients']
204   rec['servings'] = origrecipe['servings']
205   price = 0.0
206   rec['calories'] = origrecipe['calories']
207   rec['carbs'] = origrecipe['carbs']
208   rec['protein'] = origrecipe['protein']
209   rec['fat'] = origrecipe['fat']
210   rec['readyInMinutes'] = origrecipe['readyInMinutes']
211   rec['image'] = origrecipe['image']
212   rec['usedIngredients'] = origrecipe['usedIngredients']
213   for usedattr in rec['usedIngredients']:
214     length1 = len(usedattr)
215     usedattr['amount'] = round(usedattr['amount']/origrecipe['servings'],2)
216     del usedattr['originalString']
217     del usedattr['metaInformation']
218     del usedattr['unitLong']
219     del usedattr['unitShort']
220   rec['cuisines'] = origrecipe['cuisines']
221   rec['cookingInstructions'] =origrecipe['analyzedInstructions'][0]['steps']
222   reclist.append(rec)
223
224 run(app, host='localhost', port=3020, debug=True)
```