SmartGrocer: A Context-Aware Personalized Grocery System

by

Roshni Jain
BE., Rashtrasant Tukadoji Maharaj Nagpur University, 2010

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

Master of Science

in the Department of Computer Science

SmartGrocer: A Context-Aware Personalized Grocery System

by

Roshni Jain
BE., Rashtrasant Tukadoji Maharaj Nagpur University, 2010

Supervisory Committee

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

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

Supervisory Committee

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

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

ABSTRACT

Grocery shopping is a routine task that people perform to fulfill their needs for food. We suspect many people would like to do grocery shopping with the grocery list to save their money and time. While creating a grocery list, people have to follow some steps such as checking the ingredients inventory available in their homes, planning meals for few days or weeks, creating a grocery list based on their meal plan and ingredients inventory status, and looking out for deals or offers, which can be utilized in their grocery purchases. These steps can be repetitive and involve people's manual effort and a considerable amount of time to carry out effectively that makes the creation of a grocery list difficult to accomplish every time considering people's busy modern lifestyles. As many grocers begin to leverage technology, they have an opportunity to understand the relationship between the people buying behavior from their purchasing history and stores' grocery information to make profit-driven decisions and promote the reduction of food waste in stores.

This thesis presents SmartGrocer, a context-aware personalized grocery system that dynamically gathers user context including their past purchase history and budget, and store context including clearance grocery inventory that consists of those ingredients that are soon-to-expire or being on sale to recommend personalized coupons to users. The personalized coupons are automatically applied to the missing ingredients of recipes thereby reducing the recipes' cost and recommending them according to the user's food budget. Recommendation of personalized coupons to users is an

effective promotional strategy to not only saving the user's money but also promoting the reduction of food waste in stores, which eventually drives more profit to the grocery retail businesses. SmartGrocer also automates the whole process of creating a grocery list with minimal effort and time expended by the user by leveraging the user and store context.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Hausi A. Müller, my supervisor, for his supervision, encouragement, and inspiration throughout my Master's program. I am thankful to him for giving me with this opportunity and guiding me through every step and for providing me with all the support that I needed to conduct my research.

Dr. Ulrike Stege, for being my committee member and mentor, and providing feedback on this research.

Harshit and Miguel, for their valuable suggestions and helping me in devising and implementing the ideas expressed in this thesis.

Lorena, Kirti, Priya, and all Rigi and PITA group members, for their time and patience in scrutinizing the content of this thesis and ensuring it is up to proper standards.

My husband, parents and other family members, for their immense support and love, and always encouraging me to go forward.

Chapter 1

Introduction

1.1 Motivation

Over the last decade, the rise and development of technologies such as smartphones have effected significant change in our everyday lives. Numerous web and mobile applications transform our personal and professional lives, providing benefits to users with regards to portability, location awareness, and accessibility [NDA12, GS04]. Today, users can control and customize home appliances such as TVs, refrigerators, microwaves, and lighting from anywhere with their personal devices over the internet [Fre18]. The smart home concept benefits users with regards to safety and convenience, which is a boon to them considering their fast-paced modern lifestyles [Fre18, Blu18, TACH13]. Moreover, web and mobile applications allow users to perform a wide range of activities in others domains including banking and shopping. For example, the retail food industry offers many services to users to provide convenience in their busy lifestyles through applications, such as online ordering and home delivery of groceries [Kaz18]. As a result, online grocery retail has become more and more common as users gradually change the way they shop for groceries [Niu09].

The use of technology in the retail industry redefines a user's shopping behavior, thereby changing the paradigm of retail. Instead of users entering the retail environment, retailers enter the user's environment through their devices. Recent studies showed that around 79% of U.S. American users prefer shopping online [Kan18] and a regular user spends an average of 202 minutes a month on shopping applications [Ken18]. This phenomenon is attributed to using personalization techniques, which have gained importance over the past twenty years [LWM+15]. Personaliza-

tion is achieved through context-awareness, by effectively examining and adapting to changing user context, efficiently disseminating contextual information, and utilizing this information to recommend products to users [SAW94]. Companies such as Amazon[1] and Ebay[2] provide personalization to users in the form of personal welcome offers and recommendations, as well as the virtual shopping advisor to leverage their context such as browsing or purchasing histories and product likes, thus reducing browsing time on the companies' sites. Personalization improves customer satisfaction, product ratings, loyalty, sales, and the reputations of companies [WL04]. Today, grocery marketing includes brand recognition, traditional advertising, and broad in-store promotions, which are often limited as users demand more personalized shopping experiences [Ste18]. According to Lee, the future grocery ecosystem will match users with in-store products fitting their specific criteria of health, price, sustainability, and taste. This will result in a more tailored and convenient shopping experience for users [Lie18].

1.2 Problem Statement

Grocery shopping is an activity that users perform on a frequent and regular basis. It consumes a large part of their income after housing and transportation [Tip18]. Scott Hannah, president and CEO of the Credit Counseling Society, says that grocery purchases should not occupy more than 15% of a household's total gross income [Ali18]. In fact, user impulse buying results in a quick demise of their food budget [Tea18]. Thus creating a shopping list before visiting the grocery store helps in maintaining food budgets [Pla18]. While creating the shopping list, users have to analyze their current and future eating preferences, keeping in mind their health restrictions. Users sometimes search manually for recipes on food-focused online social networking services and list information about missing ingredients [Haa18]. Connors et al. mention that the cost of ingredients is the key factor in selecting recipes. Thus, if the ingredient costs are high, then users prefer to choose less expensive recipes [CBSD01], which manifests itself in the fact that user budgets plays a significant role in the selection of recipes. However, price reduction strategies such as deals and offers influence the choice of targeted recipes over others by lowering their cost relative to alternative choices [Fre03].

[1] https://www.amazon.ca/
[2] https://www.ebay.ca/

Price reduction is a marketing tactic used by businesses to entice users to use their products and services [DC05]. Grocery chains such as Walmart,[3] Save-On-Foods,[4] and Thrifty Foods[5] publish their deals and offers with the help of store flyers on a daily or weekly basis. According to a study by NCH Marketing services, 60% of users are influenced by these offers and want to utilize them in their grocery purchases [Car18]. However, users have to manually check flyers to access these offers, thereby minimizing unnecessary purchases and maximizing their savings while grocery shopping. Sometimes these offers do not meet the user's shopping needs and lack personalization. In other cases, users unnecessarily stock up on items because these items are on sale, which might, if not adequately consumed, lead to food waste and loss of money [UGS14]. Food waste is a primary concern for our society and environment [BHBR17], and does not occur only in households. A study by the Natural Resources Defense Council of U.S. shows that the big stores like Walmart and Kmart[6] throw away around 45 billion tons of food each year [BA18]. The leading causes of the waste of food items in grocery stores are excessive stock, lack of personalization, and limited knowledge of consumer behavior [BHBR17]. As a result, even after selling products at reduced prices, the substantial amount of food waste at stores persists [WHH11].

Personalization provides convenience by minimizing grocery trips, giving users more satisfaction and reducing food waste in stores [Tar18]. As per the survey from the Boston Council Group, personalization can increase grocery store revenue by 8% [Abr18]. There are many large grocery chains such as My Metro,[7] Amazon Fresh,[8] and The Fresh Market[9] that are now investing in personalization programs. These companies try to identify opportunities for personalized offers that results in success providing personalized, healthy meal planning [Eat18] and user savings in their grocery shopping [Met18]. Also, 55% of users consider personalized offers to be a prime factor influencing their choice in grocery stores [Car18]. Furthermore, the practice of tailoring offers and promotions based on a user's past purchases is likely to increase their buying activity and store sales [Gar18]. Therefore, personalized coupons for products might save money and might promote the reduction of food waste in stores.

[3] https://www.walmart.ca/en
[4] https://www.saveonfoods.com/
[5] https://www.thriftyfoods.com/
[6] http://www.kmart.com/
[7] https://www.metro.ca/en
[8] https://www.amazon.com/AmazonFresh/
[9] https://www.thefreshmarket.com/

The understanding of consumer behavior is pivotal in managing grocery store inventory and achieving personalization, which in turn helps to reduce food waste and build good relationships between the consumers and stores. Consumer behavior refers to *"the behavior that is displayed by an individual while she buys, consumes or disposes of any particular product or services."* It helps grocers to know what, where, when and how various products are being consumed [UKE18]. Thus, an inventory management strategy and knowledge of consumer behavior are critical factors to reduce food waste in stores, thereby increasing profits and satisfying user demands and needs [UGS14].

In addition, creating a grocery list while considering users' budget, tastes, and health restrictions as well as taking advantage of offers and promotions, so far still requires manual effort and a considerable amount of time to perform effectively. Therefore we need a way to automate the utilization of support (i.e., consideration of user's taste, health, and grocery offers) needed for creating the grocery list with minimal effort and time expended by the user.

1.3 Research Questions

Based on the above motivation and problem description, we formulated the following three research questions (RQ).

RQ1: How can we integrate past purchases by users and grocery product stock information to reduce the cost of personalized recipes and recommend these recipes based on user food budget?

RQ2: How can we automate the creation of a grocery list by exploiting a user's personal context and grocery store context?

RQ3: How can we promote the saving of user money and reduction of food waste in grocery stores?

1.4 Approach

To alleviate the aforementioned realistic problems and to answer the above three research questions, we propose a system called SmartGrocer, a Context-Aware Personalized Grocery System, as proof of concept for improving user grocery shopping

experiences. It also acts as a trusted agent for stores to enhance their revenue and assists in reducing perishable food waste while considering user context. SmartGrocer reduces the cost of missing ingredients of recipes which are recommended by CAPRecipes with the help of personalized coupons and recommends these recipes based on the user's food budget. In addition, SmartGrocer automates the creation of shopping lists by leveraging user context (i.e., budget and purchasing histories of groceries) and store context (i.e., grocery stock list of soon-to-expire and promotional ingredients).

CAPRecipes is a Context-Aware Personalized Recipe Recommender that acts as a third-party recipe recommendation system for our research. CAPRecipes utilizes three context areas to recommend personalized recipes to users. These are: *kitchen context* that includes attributes of the ingredients (i.e., expiry date and quantity) that are currently available in the user's kitchen, *temporal context* that includes the meal time, and *personal context* that includes likes on social media, recommended recipes list based on their own and other similar users likes, health-related information (i.e., allergies, medical conditions, calories intake), faith/belief restrictions (i.e., Vegan, Vegetarian, Ovo vegetarian), and cuisine style [Jai18].

The key features of SmartGrocer are as follows:

1. A user can create a meal plan which includes recipes for breakfast, lunch, and dinner, e.g. for a few days or weeks. Recipes on the meal plan are added by our system when a user sets the *start* and *end* date of the meal plan. After the meal plan is finalized, a shopping list will automatically be generated, including the missing ingredients for all the selected recipes. The meal plan also provides information on user savings and nutritional details for each day.

2. While creating the meal plan for the user, SmartGrocer recommends personalized coupons for some of the missing ingredients in the recommended recipes. SmartGrocer recommends personalized coupons to the user based on the information shared by the grocery store such as user purchase histories and the grocery stock of soon-to-expire products, and new products which are being promoted. With this money-saving feature, the user can now prepare recipes whose missing ingredients cost would otherwise exceed her food budget.

3. SmartGrocer provides recipe information such as the cost of missing ingredients before and after applying personalized coupons, nutrition details, and missing

and used ingredients information. Our system extracts the details of each missing ingredient, such as quantity from recipes, to compute its cost using grocery services.

4. SmartGrocer creates a grocery list which includes information of missing ingredients with personalized coupon details, if applicable and the recipes list which uses these ingredients.

5. Our system allows children to add items they need from the store to a wish list. Items from this list will move to the grocery shopping list with their personalized coupons details, if applicable after parental confirmation.

The resulting application that includes these features is a context-aware, economical application that takes user and store context into account when delivering personalized offers and automating the process of creating a grocery list.

1.5 Thesis Outline

This chapter introduced the problem addressed in this thesis. The remaining sections are organized as follows.

Chapter 2 presents a literature review on user buying behavior. It also evaluates current grocery applications and the research related to this field, explaining the need and motivation behind this work.

Chapter 3 presents the description of algorithms of the back end services that we use to achieve our system goals.

Chapter 4 presents the overall design of SmartGrocer, including the architectural and user interface design which is done on a web development framework.

Chapter 5 evaluates the system on the basis of efficiency, effectiveness and user experience by analyzing the results in different realistic scenarios.

Chapter 6 provides a summary of the work done in this thesis and presents the scope of future work.

Chapter 2

Background and Related Work

This chapter introduces the concepts important for this thesis by beginning with a literature review on consumer buying behavior that will help grocers attract more consumers to their stores and assist in providing more personalized experiences to users. This chapter also presents an overview of context, data-mining techniques for modeling context, and context-aware applications and systems, as related to our research field.

2.1 Literature Review On Consumer Buying Behavior

A consumer represents a user of a product or service [App51], and understanding her buying behavior is essential before building any e-commerce application. On the one hand, every consumer has a different outlook or attitude towards the purchase and consumption of a product [UKE18]. On the other hand, users generally follow a somewhat predictable series of steps before making any purchasing decisions, known as the *consumer decision-making process*. The consumer decision-making process starts by recognizing a consumer's need, identifying other alternatives or choices available within the market, evaluating them, making a decision based on the evaluation and lastly, assessing her own decision by analyzing her satisfaction level after making a purchase [BF15]. The consumer decision-making process is influenced by the characteristics of sellers and buyers, the environment, and the EC (Electro Conductivity) system that includes the technology and websites, as shown in Figure 2.1.

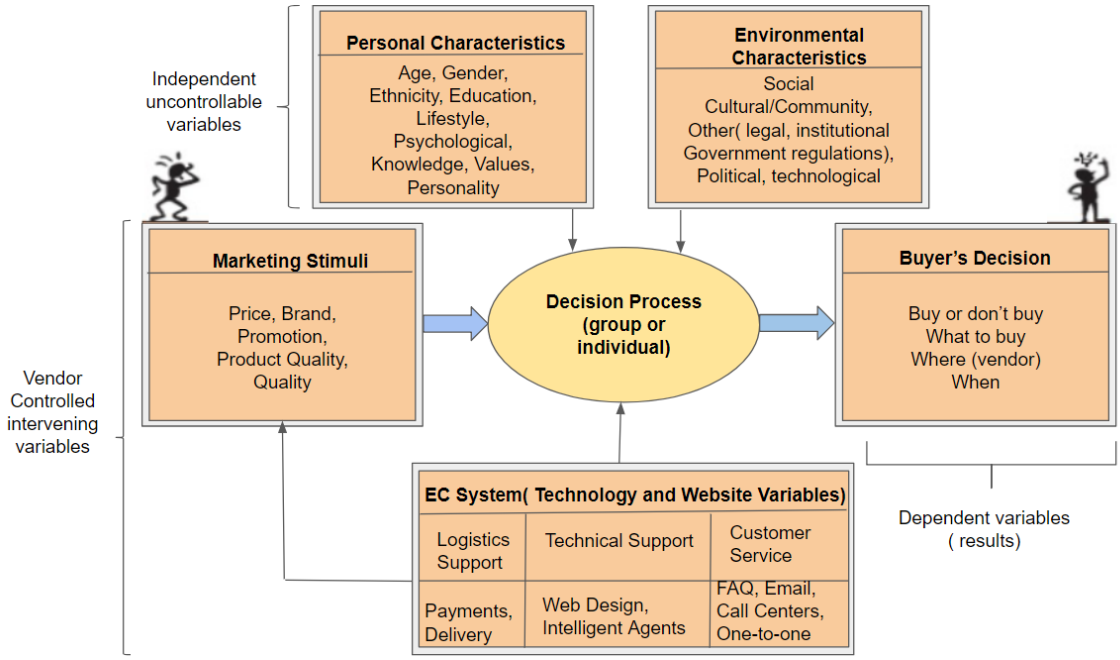


Figure 2.1: Consumer Behavior Model (Courtesy of Henry Chan Post) [1]

Grocery stores and researchers conducted studies on consumer buying behavior by examining consumer decision-making processes while purchasing grocery products [LMSR09]. This thesis reviews these studies to find out how the marketing stimuli as depicted in Figure 2.1 affects the choice of a grocery store. The literature review would assist stores in satisfying the consumer's needs, stimulating and influencing the demand for products that would result in increasing product sales and minimizing the waste of perishable items in stores. From this literature review, stores also familiarize themselves with user reactions to new products that appear on the market.

From the previous research, we found that the seven Ps (7Ps) of marketing mix — *Product*, *Price*, *Place*, *Promotion*, *Physical evidence*, *Processes*, and *People* — play a major role in determining the store choice for grocery shopping [MM11]. According to research performed by the British Market Research firm Mintel, the following attributes influence a consumer's choice in selecting a grocery store in Britain: *quality of a product*, *location*, *cheap or low prices*, *diversity in product choices*, *clean and hygienic fresh food service counter*, *length of opening hours*, and *adequate checkout facility* [MM11]. Other research performed by Engel et al. showed that *product*

[1] https://www.researchgate.net/figure/E-commerce-consumer-behavior-model_fig1_228939097

quality, *price*, *location*, *promotional offer* given on products, *store ambiance*, *seamless services* offered by stores and other factors are critical features in determining a consumer's store choice [Eng97].

In this thesis, we describe the first four Ps of marketing mix — *Product*, *Price*, *Place*, and *Promotion* — which are considered to be the element of traditional marketing mix model (i.e., 4Ps) and the 4Ps model is still widely used [Nik18]. Thus, a literature review of the 4Ps model serves a great place to start planning how to increase store sales, fulfilling the consumer's demands and needs, and reducing food waste.

2.1.1 Product

A product is defined as a tangible good or an intangible service. It acts as a medium to satiate consumer demands or needs when offered to a market for attention, acquisition, and consumption [Cle18]. A product plays a critical role in marketing management, and its mishandling might lead to heavy losses for stores [MM11]. Thus, comprehensive knowledge is needed on products before branding them to consumers. Laine showed that the product could be emphasized on three levels, where each level adds customer value and is interrelated with other two levels [Lai14], as depicted in the following Figure 2.2.

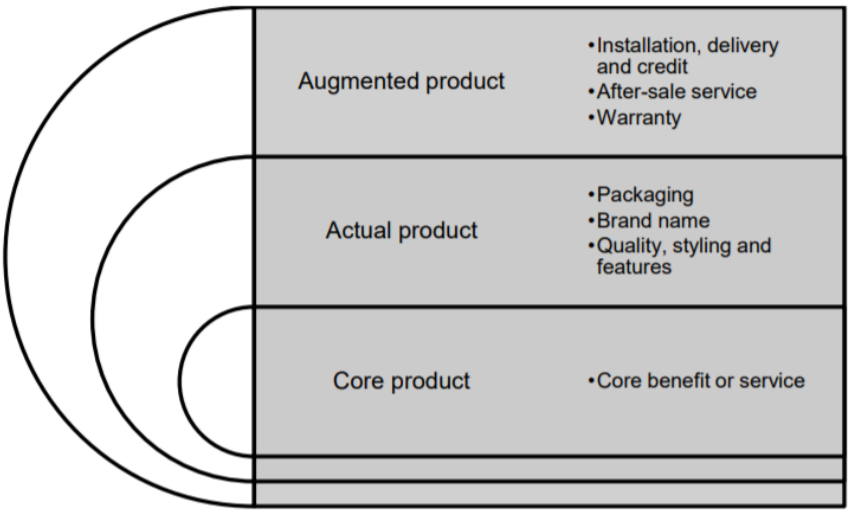


Figure 2.2: Levels of Product [Lai14]

The first layer is the *core product*, which states the worthiness of the product

regarding their benefits or services provided to consumers. The second layer is an *actual product* that turns the core benefits into a real product, i.e., generating a tangible product. This layer determines packaging details, brand name, product features, styling, and its quality level. The third layer is the *augmented product* that encompasses the core and actual level of a product and explains the additional customer services and benefits [Lai14].

In the context of grocery shopping, the determinant characteristics of the product are: food quality and a wide range of food products, which are linked to the actual level of product, and a fresh food service counter which is associated with the augmented level of product. These characteristics assist in determining consumer choice of store for their primary grocery shopping [MM11]. Moreover, grocers generally provide seasonal discounts on products to influence consumers to buy them during that particular season. However, typically consumers do not purchase all the products on sale at the same time. They typically purchase only those products that they need at that time. Thus, a long period of time is needed to study the impact of buying behavior of consumers on the selling of products, as consumers buy a substantial selection of products over a period of time. In light of this, to derive the actual effect of consumer buying behavior over the product stock, the grocer has to make his decision considering various aspects. These aspects are: what products consumers are buying at a particular moment in time, product quantity, and consumer dietary habits that can affect the choice of products and their amount [App51]. Thus, grocer decisions related to products also influence a consumer's choice of store.

2.1.2 Price of a Product

Price denotes the amount of money that consumers pay for a product or service. Good pricing strategy results in profit for the seller and value for the buyer or final consumer [MM11]. As per Lucena Matamalas et al., price is the only aspect that generates revenue while the other elements, such as product development, advertising, and promotion and selling effort involve cost [LMSR09]. Thus, price is a crucial element, and its adjustment has a profound effect on the market, producing a direct impact on revenue. It also affects the quantity sold through its influence on demand [Ivy08]. There are many pieces of evidence available from commercial and academic studies that manifest the significance of pricing in determining consumer choice of store. Muzundo et al. showed a survey performed by Buzuzi on a group

of consumers in Zimbabwe and the results explained that 90 percent of low-income earners, 88 percent of middle-income earners, and 60 percent of high-income earners placed price ahead of product attributes such as quality and brand, shop reputation and personal needs and preferences [MM11]. Moreover, many researchers depicted the following terms for measuring the significance of price, which determine the grocers' profit: attractive price, cheap/low price, and fair price. The attractiveness of prices offered by stores influences consumer choice of store [MM11].

2.1.3 Place

Place gives information on how the products can be delivered to consumers [MM11]. Place can be explained with the help of two contexts: one is related to the placing of a product in the grocery store [Inv18], and the other is the location of a store [MM11]. The location of the store has been identified as an important factor in influencing consumer choice of store, in the sense of accessibility and convenience to consumers [MM11]. Thus, we also consider place in relation to the store's location in our literature review.

In addition to location of the store, consumers also evaluate stores on the basis of product quality and price, as they can find multiple options for the same product in different stores. Consumers prefer to shop in more than one department of the store and from different specialty stores. For example, many consumers do not buy perishable products, such as fruits, vegetables, milk, and bread where they buy non-perishable products, although all these products are available in the same store [App51]. Thus, location of the grocery store location along with the product quality and price influences the consumer's choice when selecting a store for grocery shopping [App51, MM11].

2.1.4 Promotion of Products

Promotion is defined as *"the coordination of all seller-initiated efforts to set up channels of information and persuasion to sell goods and services"* [MM11]. The purpose of promotion is to reach out to the targeted market and communicate benefits of products to them effectively with the goal of increasing store sales and profits [FAH15]. This can be achieved through different modes of promotion, which are categorized as follows:

1. *Advertising* is a marketing tactic that involves non-personal promotion of ideas, goods or services through various sources of communication, such as broadcast, radio, print, and online [Lai14].

2. *Sales Promotion* is direct marketing to sway consumers to purchase a product by providing coupons, discounts, point-of-sale displays and other demonstrations [Lai14].

3. *Public Relations* relate to the creation, maintenance, and protection of the company's prestige within the market, which is achieved through building good relations with the company's public. It benefits the company by generating its goodwill (eg. a company's name, solid customer base, and good customer relations), thereby leading to an increase in product sales. The communication tools used to make good public relations are press conferences, public engagement, press releases, and community service programs [fB18].

4. *Personal Selling* is a medium to showcase the company's reputation within the market through personal representation by the company's sales force with the help of events and conferences [Lai14].

5. *Direct Marketing* involves interacting with consumers to receive an immediate response and to cultivate long-lasting consumers with the help of catalogs and personalized offers sent by mail (for example, flyers of store deals or offers and money-saving packages) [Lai14, fB18].

The different forms of promotional activities, such as customer loyalty cards, discounts or sales promotions on grocery products, and proper advertisement influence consumer choice of grocery store [MM11].

Therefore, from the literature review, we conclude that the 4Ps model of the marketing mix are directly or indirectly affecting the consumer's choice of a grocery store. Detailed evaluation will help grocers make their business profitable and provide satisfaction to their customers.

2.2 RESTful Web Services

Roy Fielding first introduces REST in his academic dissertation, *Architectural Styles and the Design of Network-based Software Architectures*, in 2000 at the University of

California [Rod18, FT00]. He analyzed a set of software architecture principles of distributed computing that uses the web as a platform. Now, years after its introduction, REST has become a critical technology for web and mobile applications [Vaq18]. The following principles that together encourage a concrete implementation of the REST web service in the application thereby making it lightweight, simple, and fast [Ora18].

1. *Resource identification through URI*

   Resources, such as image, web page and business information can be accessed by referring to their address. The *Universal Resource Identifier (URI)* is the global addressing space for identifying resources and services. A RESTful web service uses directory structure-like URIs. The directory structure-like URI follows a hierarchical pattern whose root points to a single path from where different subpaths are created and used to expose the service's primary areas [Rod18, Vaq18, Ora18]. For example, in a discussion threading service that allows user discussion on various topics and access to information on any topic, the URI can be defined as:

   ```
   http://www.discussionforum.org/discussion/topics/{topic}
   ```

   Here, the root is */discussion*, which has a subpath */topics*. There is series of topics, such as technology and marketing under the */topics* path, each of which is pointing to a separate discussion channel. The URI does not provide information about the action or operation for example, GET or POST operation, which can only be achieved through HTTP methods [Rod18, Vaq18, Ora18].

2. *Uniform Interface*

   Resources are manipulated using a fixed set of *HTTP* methods explicitly. REST establishes a one-to-one mapping between the Create, Read, Update, and Delete (CRUD) operations and *HTTP* methods as illustrated below:

   POST - mapped to Create/Update operation that creates a new resource or also updates the state of an existing resource on the server
   GET - mapped to Read operation that retrieves the current value or state of the resource from the server
   PUT - also mapped to Create/Update operation that updates the state of the

existing resource
DELETE - mapped to Delete operation that removes the resource.
The CRUD operations create, update or delete resources on the server/cloud side based on the action request received through the HTTP requests.

3. *Statelessness*
   The Statelessness property simplifies the design and implementation of the web service request as the request messages are self-contained. These requests do not use external information, such as session cookies and state of the server to achieve synchronization between the server data and the application, thereby improving the performance of web services. For example, when we use Twitter [2] from our mobile phones and check for recent direct messages, we utilize the stateless service where the response is entirely independent of any server state, and everything is stored on the client's side (i.e., on our mobile phones) in the form of a cache [San18].

4. *Self-descriptive messages*
   The resource representations provide information about the current state of the resource at the time the application requests it. The resource representations can be seen in various formats, such as XML, JSON, and HTML. The Extensible Markup Language (XML) and JavaScript Object Notation (JSON) are widely used resource representation formats. XML is a general purpose text-based structured data specification, which shows the response with the help of XML tags while JSON shows the response in the form of key-value pairs [Res18].

Following the REST principles, we use two REST API services in our research which are Spoonacular [3] and Foodpairing [4] API to achieve our system goals.

2.3 Context-Aware Computing

During the last few years, the term *context-aware computing* is being used more frequently in relation to the use of modern technologies and brings users one step closer

[2] https://twitter.com/?lang=en
[3] https://spoonacular.com/food-api
[4] https://www.foodpairing.com/en/home

to the pervasive computing vision [SAW94], a vision of creating an environment saturated with computing and communication capability and integrated with human users [Sat01]. Context-aware computing applications are characterized as the ability of the applications to adopt the changes of the environment in which they are run [SAW94]. Context-aware computing applications first understand the context and the reason behind the current situation to perform a suitable operation. This can be achieved by autonomously updating the reactions to changes in user context over a period of time, which is referred to as implicit Human-Computer Interaction (HCI) [SLND12] since the computer responds based on implicit user activity (for example, moving from one location to another). Before implementing the system that works on implicit HCI, we need to elucidate various concepts related to the context-aware computing field. These are: what is context, what the different context modeling approaches are, and how data mining techniques are useful in modeling the contextual information.

2.3.1 Context

Context is defined by many researchers and is continuously modified and refined to make it more applicable to computer applications. Researchers mainly focused on two aspects while defining the context, a *user-centered* focus, and a *system-centered* focus. Schilit et al. defined the user-centered understanding of context applied to the field of mobile computers as the relationship between the user and the different parts of the situation, such as where you are, who you are with and what resources are nearby [SAW94]. They considered context as more than just information about the location of a user, since there are other factors too that are mobile and continually changing and are of interest to the user. Lieberman et al. defined the context more technically and centered on applications instead of the user [LS00]. They stated that context is any information that the system senses but not the explicit input and output, and which affects the system computation. The explicit input is the user inputs given in the form of command via the Graphical User Interface (GUI) of the application, and the output is the outcome of the system.

The latest definition by Dey that took the above two aspects of context into account and defined the context in a more general manner, is the following:
"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the inter-

action between a user and an application, including the user and applications themselves." [Dey01]
This definition is widely used and is related to every element of the context, including the user and the application that produces a direct effect between their interaction [BHS04]. Therefore, the latest definition of context is apt for this research because we are using the user's context, such as their shopping behavior, a store's grocery stock, and the SmartGrocer response, i.e., personalized coupons to help users buy personalized groceries at discounted prices.

2.3.2 Grouping of Context Variables

Researchers explain context with the use of different context variables. These are: *Location*, *Time*, *Identity*, *Environment*, *Social Setting*, *Network*, *Season*, *History*, *Task/Activity*, and *Device* depicted in the Table 2.1. They group these context variables into different categories. The grouping provides clearer understanding about the context while creating any context-aware application.

Schilit et al. also grouped context variables by their situational questions, which are: "where you are", "who are you with", and "what resources are nearby" [SAW94].

Table 2.1: Context variables

	Rahlff et al. [RRH01]	**Brown et al. [BBC97]**	**Schilit et al. [SAW94, ST94]**	**Hess et al. [HC03]**	**Dey et al. [DAW98]**	**Beale et al. [DAS01]**
Location	✓	✓	✓	✓	✓	
Time	✓	✓		✓		
Identity	✓	✓	✓	✓		
Environment	✓	✓	✓	✓	✓	
Social Setting			✓			
Network			✓			

Schmidt et al. grouped the context variables into two categories, as shown below [SBG99]:

1. *Human factors*, such as user, social environment, and task.
2. *Physical environment*, such as location, condition, and infrastructure.

Dey et al. grouped the context variables into three categories [DAS01]:

1. **Human factors**: which include
   (a) Information about users, such as habits, physiological conditions, and emotional state.
   (b) Users' social behavior, i.e., social interaction with other people or group dynamics.
   (c) Users' tasks, i.e., spontaneous activity, general goals.
2. **Computing factors**, such as network connection and user interface size.
3. **Physical context**: which includes
   (a) *Active context*: context variables that influence the behavior of the application autonomously based on the sensed information.
   (b) *Passive context*: context variables that act as secondary information that is relevant but not critical.

2.3.3 Context Modeling Approaches

When we have a large set of contextual data available from different sources, it can be challenging to handle and interpret it in a precise and traceable manner. There are various approaches to modelling contextual data with the goal of having a proper context-management system. These are as follows:

1. *Key-Value models*: This type of model is frequently used in distributed service frameworks. It models the contextual information in the form of key-value pairs where the key is any context variable, for example location, and the data associated with the key is stored as a value [SLP04].
2. *Markup Scheme models*: This type of model is used to hide the complexity of contextual information that occurs when dealing with the dynamicity of context. This modeling approach represents contextual data in the form of profiles, consisting of markup tags with attributes and content [SLP04].

3. *Graphical models*: This modeling approach uses the Unified Modeling Language (UML) to model the context. The annotation of the UML diagrams describes which information is related to the specific context. This kind of approach is beneficial for modeling the relational database in an information system based on a context management architecture [SLP04].

4. *Object Oriented models*: This modeling approach hides the details of context processing through an object-oriented technique — encapsulation or re-usability on an object level. The contextual information can be accessed through specified interfaces only. This approach is used to simplify knowledge in very complex domains and systems where the problem arises due to the dynamics of the context [SLP04].

5. *Logic Based models*: This modeling approach derives the concluding expression or fact from the sets of other expressions or facts based on certain conditions (i.e., logic). The context in this type of models consists of rules (i.e., *if-then* statements), facts, and expressions [SLP04].

6. *Ontology Based models*: This kind of approach models the context with ontologies that define the inter-relation between the context variable and the information associated with it. The system based on these models provides a set of ontologies to characterize context variables, such as person, place or other variables within their context [SLP04].

In this research, we use Key-Value models to model the personalized coupons and personalized recipes in the form of key-value pairs. The Logic Based modeling approach is used to derive the information of frequently purchased ingredients from the transaction details of users to recommend personalized coupons to them.

2.3.4 Data Mining Techniques for modeling Contextual Information

The goal of any context-aware system is to provide users with what they need, without asking them explicitly, through the use of personalization. For example, in the context of web browsing, personalization implies the delivery of dynamic content to a particular user, such as links, advertisements, and product recommendations from the results obtained after modeling her contextual data [Mob07]. Over the years,

researchers proposed different data mining algorithms to model the contextual information for achieving personalization from the system. This section reviews some of the data mining techniques to model the contextual information that is closely related to our research.

Association Rules

The association rules algorithm is a data mining technique that is widely used in real-world applications for determining frequent co-occurring associations between seemingly unrelated data [RG08]. It assists in making effective decisions in the business field. For example, in our research, the management of extensive collections of products in grocery stores requires some key marketing decisions, such as what products to put on sale, how to design user-centric coupons to maximize stores profits, reduce food waste within stores, and provide maximum user satisfaction. There are other areas as well where association rules can be applied, including library circulation data, the study of protein composition, and the study of economic census [RG08].

The association rules algorithm produces a set of rules in the form of *if-then* statements when applied to the complete user's transaction database. The antecedent, i.e., the *if* part, and the consequent, i.e., the *else* part, is the set of items (also known as itemset) that are disjoint (i.e., do not have any items in common). These rules are obtained depending upon the value of the following three parameters, which are described below:

1. *Support*: Support is the number of transactions that include all items in the antecedent and consequent parts of the rule [RG08].

2. *Confidence*: Confidence is the ratio of the support value and the number of transactions that include all the items in the antecedent [RG08].

3. *Lift*: Lift is the ratio of the confidence and the expected confidence set by the user [RG08].

Next, we explain the association rule for mining supermarket data.
Let I =$\{i_1, i_2, i_3....i_d\}$ be the set of all items available in the supermarket database and T = $\{t_1, t_2, t_3....t_N\}$ be the set of all transactions related to a single user. Each transaction t_x consists of a subset of items from I. An item set Z is the subset of any transaction t_z. For example, let's say in a particular transaction the items purchased are {Apple, Milk, Bread, Tomato}, then Z can be, for example {Apple, Milk} or

{Apple, Bread, Tomato}. We first generate the frequent itemsets by searching all possible subsets (i.e., Z) based on the minimum support threshold value [TSK05]. Then, we use expected confidence value which is set by the user to these frequent itemsets in order to form rules. A rule is defined as an implication of the form X →Y, where X and Y are antecedent and consequent itemsets, respectively. So, if a user buys all the products of X, then he can buy the products of itemset Y, which is the property of association rules. The strength of the rule can be measured based on the support and confidence values, which are computed as given in [TSK05]:

$$Support, s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

$$Confidence, c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

where $\sigma(X)$ is the support count, which refers to the number of transactions that contain a particular itemset and is calculated as in [TSK05]:

$$\sigma(X) = |\{t_x | X \subseteq t_x, t_x \in T\}|$$

Information Filtering

Information filtering is a technique that recommends to users relevant information from a large collection of valuable data from different computer network sources. For example, an application recommends items to a user based on her profile, which is created through an information filtering process that includes a list of items liked by her. We discuss two ways to measure the actual relevance of an item provided by the system to a user, called *explicit* and *implicit* user feedback. For explicit feedback, the user is asked to express the degree of satisfaction achieved after using the recommended item through feedback which is analyzed with the text analysis methods to understand the user's preferences [JSK10]. For implicit feedback, the relevancy of the item is determined by observing the user's behavior which includes reading time, saving, printing, or selecting internet news articles, movies, books, or grocery items [KT03]. Based on the feedback, the user's profile is updated accordingly for future recommendations. There are two main approaches in Information Filtering: Content-Based Filtering and Collaborative Filtering [SMS08].

1. **Content-Based Filtering**

   Content-Based Filtering computes the similarity between the user profile and the item profile of all items. The user profile consists of user preferences and needs, which are derived either explicitly (e.g., through likes, ratings on items) or implicitly (e.g., through a user's behavior like how frequently a user buys an item which is determined from their previous purchases). The item profile contains a set of attributes of items [Das18]. The system recommends items based on the similarity coefficient, which may satisfy user needs or tastes. The similarity coefficient is calculated as [LCL12],

   $$\text{S(c,i)} = \text{score(ContentBasedProfile(c), Items(i))}$$

2. **Collaborative Filtering**

   In Collaborative Filtering, items are recommended either based on the similarity coefficient of the user to other users who liked similar items in the past, known as *User-User Based Collaborative Filtering*, or the similarity coefficient of the items (eg. grocery products) with other items, which are liked by a fixed group of users, known as *Item-Item Based Collaborative Filtering* [SKKR01].

   For example, according to one of the scenarios related to our research, we want to determine which ingredients can be paired well. As per typical Collaborative Filtering scenario, we can apply the *Item-Item Based Collaborative Filtering* by calculating the cosine similarity between the attributes of two ingredients, e.g., the smell of the two ingredients as shown below [SKKR01]:

   $$sim(i,j) = cos(\vec{i},\vec{j}) = \frac{\vec{i}.\vec{j}}{||\vec{i}|| * ||\vec{j}||}$$

   Where i and j are the aroma molecules values of the two ingredients.
   We can calculate the cosine similarity for other ingredients. Then, ingredient pairs whose similarity value greater than the similarity threshold (η), pairs well and can be used in the preparation to get different flavors, which might be delicious in taste. The η can be varied to get the better pairing results. Note that an ingredient can be paired with many ingredients.

2.4 Context-Aware Systems

Context-Aware Systems pervade everyday life by offering relevant information at the right time in the right place to the right person [Fis12]. SmartGrocer appropriately satisfies user needs while grocery shopping by leveraging the available resources, which are user context (i.e., budget and previous transactions details) and grocery store context (i.e., ingredients pricing and a list of soon-to-expire and promotional ingredients). Also, SmartGrocer assists grocery stores in promoting the reduction of food waste in stores, thus making our system context-aware. Below, we review a few context-aware grocery applications and systems that are already implemented for user convenience, highlighting their contextual information.

2.4.1 Context-Aware Grocery Applications

Over the last few years, the pattern of grocery shopping has changed from traditional brick-and-mortar store shopping to online. Now, more and more tech-savvy grocers publish their context-aware grocery apps, such as Whole Foods,[5] and My Metro[6], which can offer much more than just grocery list-making capabilities. Examples are scanning of products, instant delivery, selection of recipes with the ability to add ingredients to the list, digital grocery coupons, and rewards points [Sta18, Fis18]. Following this, we review two grocery applications with regards to their functionality, shopping features, and benefits along with their contextual details.

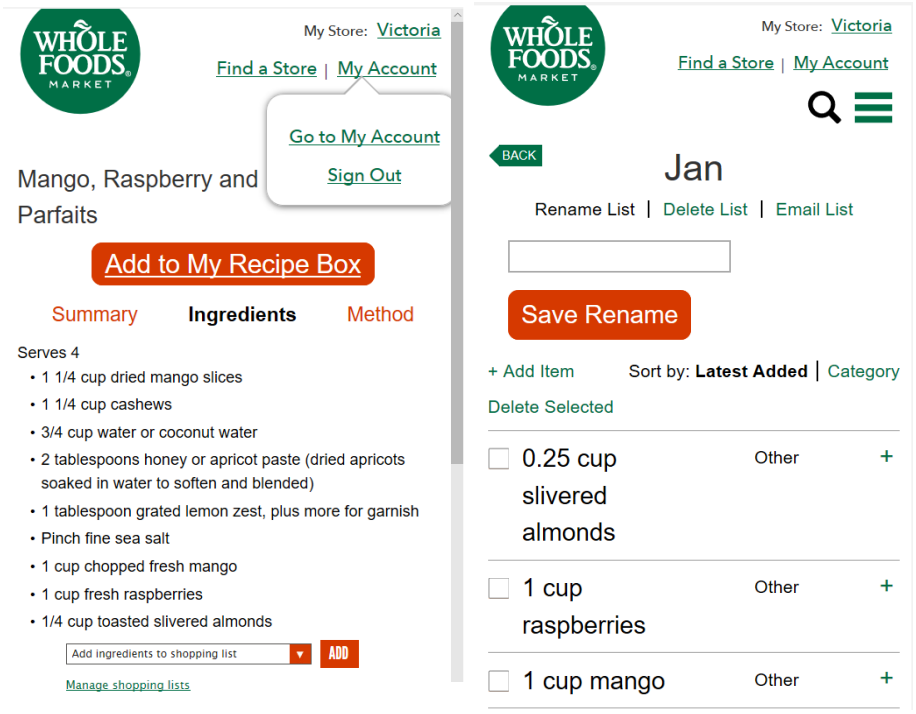
Whole Foods

Whole Foods supermarket has a web-based and mobile (both iOS and Android based) application that provides an option for users to add all the current week's sale items, or any item shared by the family members via text or email only, directly to their grocery list. The application has a separate recipe section, allowing users to plan their menu on the fly and add any or all ingredients of the selected recipes to their grocery list depicted in Figure 2.3 that sync across devices [Epi18].

Context-Aware features: The application uses ingredient details of a recipe and general coupons (i.e., based on grocery stock information of the store and grocer's decisions) for products while creating a grocery list for the user. It also features *local*

[5] https://www.wholefoodsmarket.com/
[6] https://www.metro.ca/en

sales, which has information about location-based product offers. For example, an offer for tomatoes in the New York region that is not applicable in other regions such as Victoria, BC.



(a) Whole Foods recipe instruction section

(b) Whole Foods shopping list section

Figure 2.3: Steps showing the addition of recipe ingredients to shopping list [Epi18]

My Metro

My Metro app is a mobile application (both iOS and Android based), which saves the user time and money while shopping. This application recommends recipes according to lifestyle and provides the option to add ingredients directly to their shopping list. The app has a separate section, known as *Just for me* and shown in Figure 2.4 that aims to save the user money and provide convenience during grocery shopping with the help of multiple app features as explained below [Met18]:

- By offering personalized coupons on items that users buy frequently. These coupons save expenditure in the form of reward points depicted in Figure 2.5a.

- By creating a personalized flyer that organizes deals according to user taste, which they selected from the general flyer; and,

- By providing comfort while creating a grocery list with the help of *My usuals* section, which stores the user's frequently purchased items information.

It also rearranges the shopping list by categorizing items according to store sections as shown in Figure 2.5b, which speeds up grocery shopping [Met18].

Context-Aware features: My Metro application uses a user's past purchases and their general profiles to recommend personalized coupons, and promotes adding frequently purchased items to the shopping list.

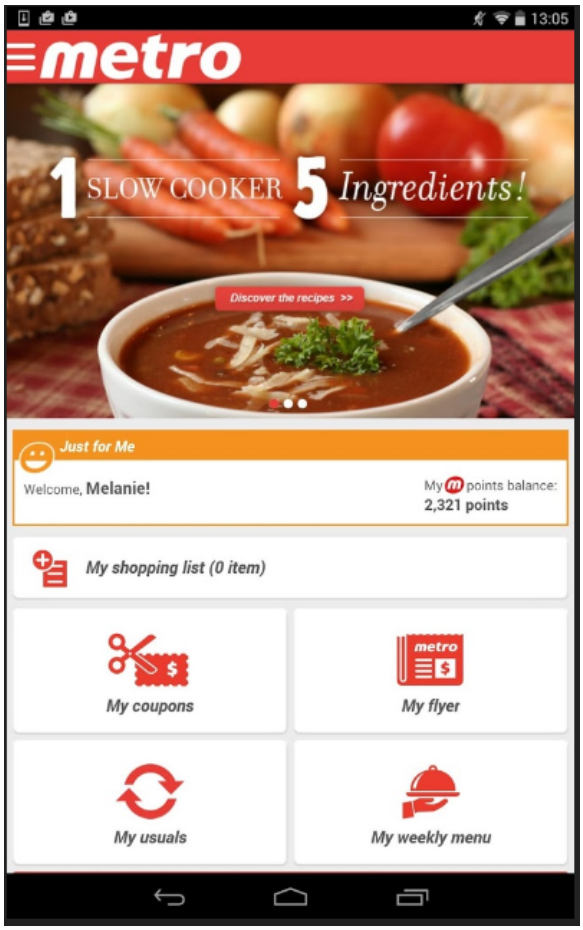
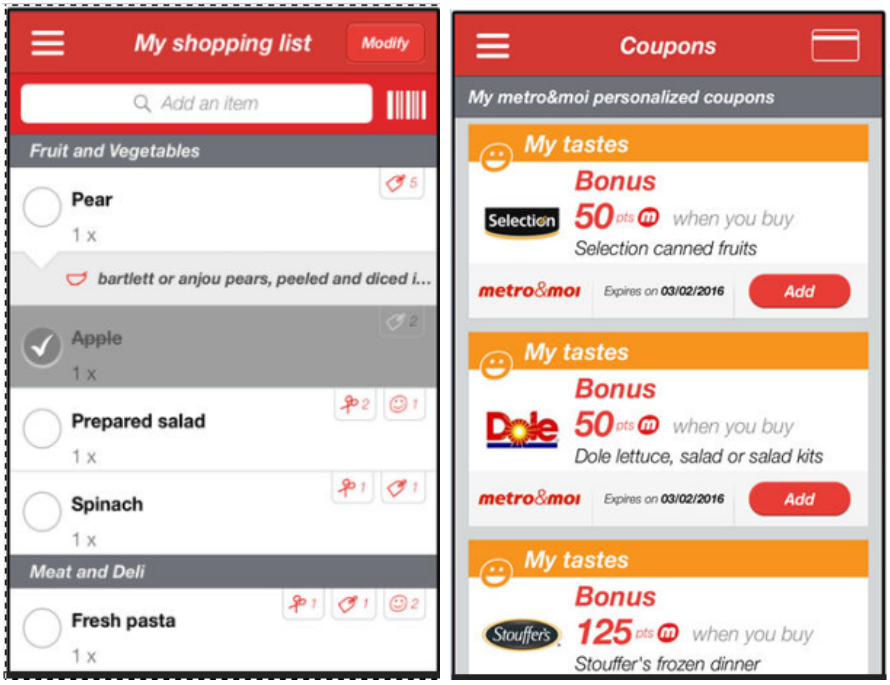


Figure 2.4: My Metro application [Met18]



(a) Shopping list with items categorized in different sections.

(b) My Metro personalized coupons

Figure 2.5: Screenshots of MyMetro features [Met18]

Note that the My Metro considers user context (i.e., past purchases of a user) for recommending personalized coupons whereas we include the store's grocery stock information and user's previous purchases to recommend personalized coupons from SmartGrocer. However, we could incorporate other contexts in SmartGrocer such as other user's context, season, and the information related to placing of products within the store for recommending coupons that might assist in increasing store's sales.

2.4.2 Context-Aware Grocery Systems

In addition to numerous context-aware grocery applications available on the market, researchers discovered context-aware systems in the retail grocery field to make user shopping easy and convenient while considering different aspects of grocery shopping. Following this, we review prior work done by researchers to improve the grocery shopping experience, which is explained in the following section.

iGrocer

iGrocer is a ubiquitous, pervasive smart grocery shopping system, which helps users shop healthy and makes their shopping trips convenient and quick. iGrocer was developed using the MotoSDK wireless toolkit and was tested on Motorola's 185s iDEN based mobile phone with the barcode scanner for research purposes. This system provides various approaches to create a grocery list, which relieves users from the pain of doing it manually and conveniently plans their shopping trips around their other commitments. These approaches are as follows [SNH03]:

1. *Personalize Categories*: Users have the option of personalizing their shopping list by selecting all those categories from the available categories, which they would typically go through to find their grocery items. For example, a user may buy her cosmetics from the mall instead of a grocery store. Then, with iGrocer, the user has an option of not choosing the cosmetic category while customizing her shopping categories available from the particular store. So, next time when she wants to add an item to her grocery list, she is presented only with the chosen categories. Also, users can create their own categories to further reduce time in browsing items from the selected categories. For example, the user can create a category for the top five grocery items that she buys most frequently, naming it as *Everytime items*.

Figure 2.6: Customize category feature of iGrocer [SNH03]

2. *Surf Category*: This is a category included in the iGrocer system, which has a list of items compiled based on users' past purchases and new food items at the grocery store. So now users are just two clicks away from the exact item they want, or from trying some new items for a different taste.

3. *By Recipe*: Users can also create their shopping list from the recipe instructions by selecting all the ingredients of a recipe. Recipes recommended by iGrocer are limited according to the user's health constraints.

4. *By Scanning Item*: Users can add items to their shopping list by scanning the barcodes of those items which are finishing soon, depicted in the Figure 2.7. Also, users can save the grocery coupons, which are useful to them in the same manner.

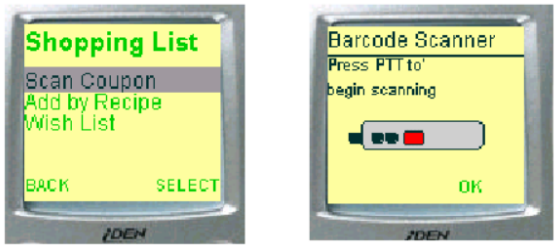


Figure 2.7: Scan coupon/item feature of iGrocer [SNH03]

The main work of this system is that it has a feature that arranges the final grocery shopping list according to aisle number and relative location in the aisle as shown in Figure 2.8. With this feature, users can perform quick shopping by traveling the shortest path to pick up all the items mentioned on their shopping list [SNH03].

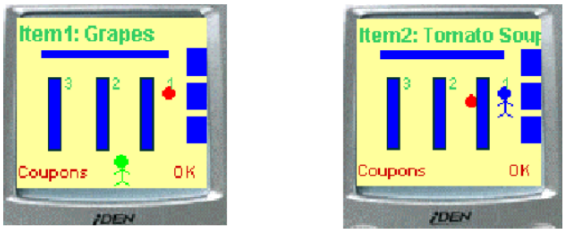


Figure 2.8: Aisle map showing location of items in the store [SNH03]

Agent-Based Grocery System

Agent-Based grocery systems automate the grocery shopping process by utilizing user preferences, grocery stock data, and store information. The system has two agents, known as the *buyer agent* and the *store server agent*.

The *buyer agent* manages the preference knowledge of users, grocery stock data, store information, discount sales information, and recipe knowledge. The *buyer agent* is divided into two agents: *user agent* that handles the interaction between users and SSA, and the *information management agent* that shares information with the *user agent* to assist buying activity. The *information management agent* organizes and manages all the information gathered from a user's grocery preferences, recipes, grocery stock, store information, and discount sales information, thereby helping create a grocery list for users.

The *store server agent* gathers grocery information from several store server agents, and provides this information at the request of *user agent* or *information management agent* [JKS00].

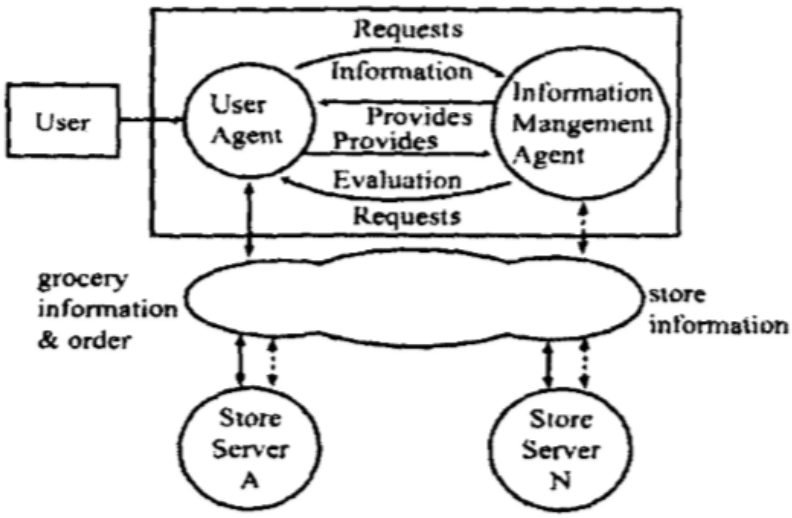


Figure 2.9: Overview of an Agent-Based grocery shopping system [JKS00]

As we can see from the previous sections, grocery applications and systems can have an option for users to select recipes and coupons, which are either general or personalized, from their system or app. Also, these systems automatically create a grocery list based on user selection of either recipes or discount coupons of ingredients, but

the selection of recipes and coupons are not being considered together when creating a grocery list.

Moreover, developers and researchers of the above mentioned apps and systems do not consider a user's past purchases, or the store's stock of ingredients that are expiring or on sale, when generating personalized coupons. Users and grocery store context are considered to be crucial factors, as argued in Chapter 1, to promote saving money and reducing food waste. Furthermore, the process of using personalized grocery coupons and user budget for recipes, selection of recipes, and creation of a grocery list based on the recipe selected, has not been integrated, which is the goal of SmartGrocer.

2.5 Summary

This chapter presented a literature review on consumer buying behavior to illustrate the significance of 4Ps model of marketing mix that influences user choice when selecting a store for their grocery shopping. This chapter also explained the RESTful web services and its main properties. We discussed the different aspects of context-aware computing, such as the definition of context, grouping of context variables, and the models and data mining techniques used to interpret context to produce a proper context-management system and achieve personalization. Finally, we explored context-aware grocery applications and systems available on the market, which assist users to achieve a quick and convenient grocery shopping experience.

Chapter 3

SmartGrocer Backend Services And Algorithms

This chapter presents the implementation details of the backend services and algorithms used to achieve goals of SmartGrocer as described in Chapter 1. In this chapter, we explain RESTful web services used in our work through pseudo-codes of algorithms. We also describe the *POS Tagging Mean Probability* algorithm, used for matching ingredient names retrieved from third-party applications with ingredients available in the grocery store database, and *Personalized Coupon Engine* algorithm, used for providing personalized coupons to users. For completeness, the complete source code for all the algorithms is available in Appendix A.

3.1 Foodpairing API service

The Foodpairing service provides a method to determine combinations of food and drinks that can go well together [Foo18b]. We use this service in our research to recommend ingredients at a discounted price from the soon-to-expire and promotional ingredients list provided by the grocery store, which can be paired/combined with the user's frequently purchased ingredients. Users might have different recipe choices in their recommendations from CAPRecipes in the future. Also, it helps grocery stores to quickly sell the maximum number of ingredients from their soon-to-expire ingredients list, thereby promoting the reduction of food waste in stores and increasing the store sales.

3.1.1 Science Behind Food Pairing

Researchers of the Foodpairing API found that five senses of humans assist in identifying different food flavors, such as salty, sweet, bitter. These are: *sight* (i.e., through the impact of color or presentation of food), *hearing* (i.e., through the expectation of sound), *taste* (i.e., through flavors of food), *touch* (i.e., through physical contact with food), and *smell* (i.e., through odor of food). Out of these five senses, they found that in 80% of the cases, smell is the key driver and is crucial for the synergy of food and drinks. Moreover, they discovered that humans can differentiate up to 10,000 different odors through their sense of smell. Odors consist of one or more aroma molecules, which can be perceived both through nose and mouth, and are volatile [Foo18b].
To find pairings, the Foodpairing team discovers aromas relevant to human smell from the aroma profile generated from the *Gas Chromatography coupled Mass Spectrometry (GC-MS)* [GW16] of an ingredient, for example strawberry as depicted in Figure 3.1.

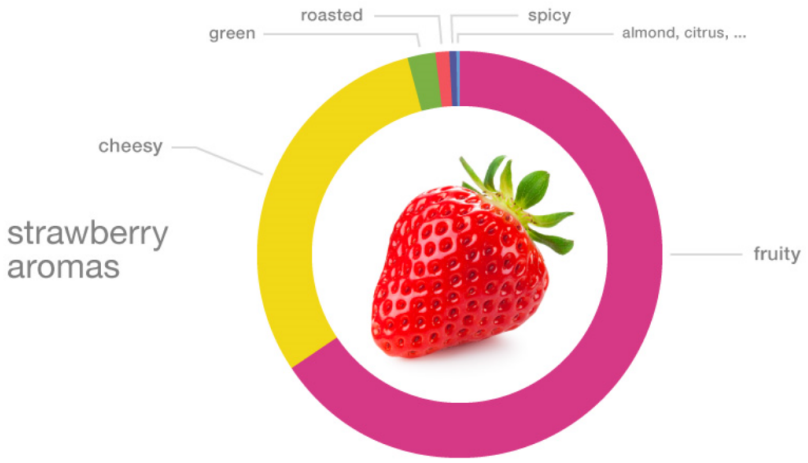


Figure 3.1: Aroma profile of strawberry relevant to human smell [Foo18b]

From the above figure, the Foodpairing team determines that strawberry has different aromatic smells, such as cheesy, fruity, spicy and so on. In the same manner, the team generated the aromatic profile relevant to human smell of other ingredients as well. After determining the aromatic profile of most of the available ingredients, the Foodpairing team created an algorithm, using scientific techniques such as data analysis and machine learning to calculate the similarity between aroma molecules in a manner depicted in Figure 3.2. Foods that maximum aroma molecules are more likely to pair [Foo18b]. For example, from Figure 3.2, the Foodpairing team found that the roasted flavor of strawberry matches with the roasted flavor of chocolate

and, based on this result, the team deduced that chocolate is one of the ingredients with which strawberry can be paired well [Foo18b]. Strawberry can also be paired with other ingredients, such as Parmesan or Basil depicted in Figure 3.2, based on the similarity of other aromatic flavors.

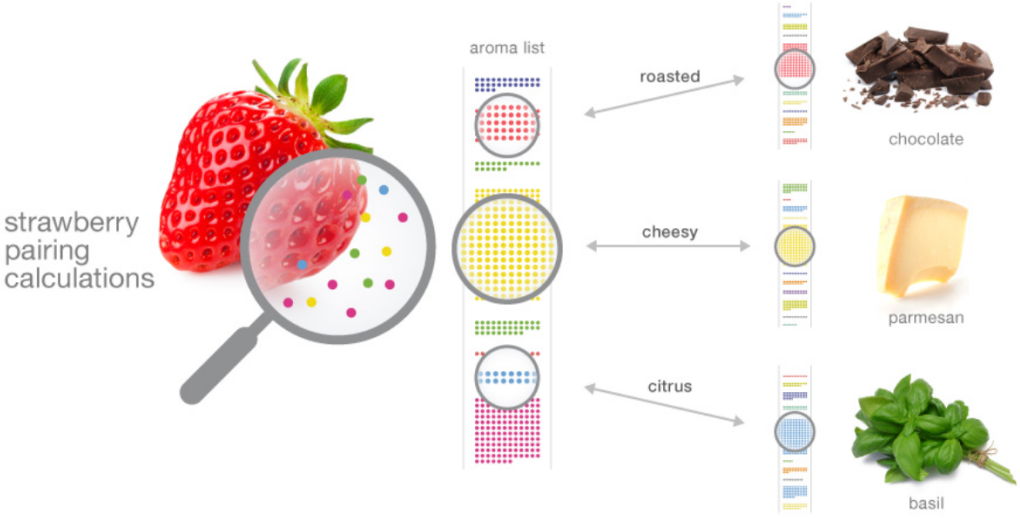


Figure 3.2: Comparison of strawberry aroma molecules with other ingredients aroma molecules [Foo18b]

3.1.2 Foodpairing API Methods

The Foodpairing team builds API requests for retrieving different information of ingredients, such as ingredient nutrients, ingredients pairing, and categorization of ingredients. In our research, we are using API calls for ingredient details, such as their IDs and pairing results, which are illustrated in the next sections:

Ingredients Information

In order to find the pairing information for any set of ingredients, we have to first determine their *id*s from the Foodpairing ingredients database using *Search Ingredients* request endpoint, as given below [Foo18a]:

```
GET https://api.foodpairing.com/ingredients?q=<set of ingredients>
```

The required parameters to execute this request are: set of ingredients separated by comma and the service API key.

A successful *200* HTTP response will return with the JSON output of the relevant match, which includes, but is not limited to, ingredient details such as *id*, *name*, *product*, and *description*. For example, the code snippet depicted in Listing 3.1, shows the JSON response of strawberry:

```
[
  {
    "id": 129,
    "name": "Strawberry",
    "product": "Strawberry",
    "description": "The strawberry is a member of the rose family. "
  },
  {
    "id": 5842,
    "name": "Strawberry 'Calinda'",
    "product": "Strawberry 'Calinda'",
    "description": "Calinda is a unique new strawberry for the winter
   season."
  }
]
```

Listing 3.1: Simplified view of the response from Search Ingredients request

Then, we use the Part-Of-Speech (POS) Tagging Mean Probability algorithm (c.f., Section 3.2) to find a perfect or most relevant match of queried ingredients. Here, the queried ingredient is strawberry, and from the JSON output as depicted in Listing 3.1, we see that the most relevant match is *Strawberry*. Based on the matching results, we determine a list of *id*s of the most relevant matched ingredients.

Ingredient Pairings Information

The pairing information for the set of ingredients can be determined with the use of their *id*s extracted from the response of the *Search Ingredients* request and the service API key through the following request endpoint as shown below [Foo18a]:

```
GET https://api.foodpairing.com/ingredients/<ingredient ids>/pairings
```

The HTTP *200* response is a JSON output that includes each paired ingredient's relative score (i.e., *rel*) and match score (i.e., *abs*), and ingredient details of paired

ingredients, such as *id*, *name*, and *description*. For example, the API request endpoint and pairing results of strawberry are shown below:

```
GET https://api.foodpairing.com/ingredients/129/pairings
```


```
[
  {
    "matches": {
      "all": {
      "rel": 0.4542495608329773,
      "abs": 0.9661538722888957
      }
    },
    "_links": {
      "ingredient": {
      "id": 1376,
      "name": "Butter (fresh)",
      "product": "Butter"
      }
    }
    },
    {
    "matches": {
    "all": {
    "rel": 0.4122965633869171,
    "abs": 0.8769230739975117
    }
    },
    "_meta": {
    "index": 2
    },
    "_links": {
    "ingredient": {
    "id": 1518,
    "name": "Dark chocolate",
    "product": "Dark chocolate"
    }
    }
  }
]
```

Listing 3.2: Simplified view of the response from Ingredient Pairings request

The pseudo-code below describes an approach we applied when using the Foodpairing API service to generate a list of paired ingredients based on the user's frequently purchased ingredients. The full source code is in Appendix A.2.

Algorithm 1 Paired Ingredients List based on Foodpairing Service

```
Query the Search Ingredients API request using frequent ingredients list
Set id_list = list()
Set counter = 0
Set abs_score_threshold = 0.8                ▷ Setting similarity threshold value to 0.8
for each ingredient in frequent ingredients list do
    Extract id of most relevant match from the Foodpairing Search Ingredients
    response using the POS Tagging Mean Probability algorithm
    id_list[counter] = id
    counter = counter + 1
end for
Create an id string, ids_string = ",".join(id_list)
Query the Ingredients Pairing API request using ids_string
Set paired_list = list()
Set counter = 0
for eachPairingIngredient in the Foodpairing ingredients pairing response do
    if eachPairingIngredient[abs] > abs_score_threshold then
        paired_list[counter] = eachPairingIngredient
        counter = counter + 1
    end if
end for
return paired_list
```

3.2 Spoonacular API

The Spoonacular API is the food and recipe API that allows you to access a great resource of recipes, ingredients, grocery products, and restaurant menu items to build food-related applications. The Spoonacular team uses food ontology as demonstrated in Figure 3.3 to construct semantic relationships between recipes, ingredients, and store-bought products that help individual app developers provide tailored food-related recommendations such as recipes and grocery product details to users. The Spoonacular API has information for more than 2600 ingredients in their

food database, which was sourced from the United States Department of Agriculture (USDA) [Spo18].

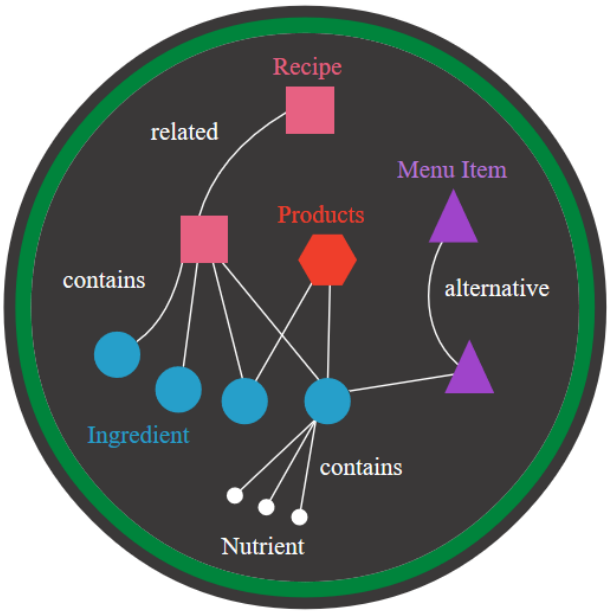


Figure 3.3: Food Ontology [Mar18]

In our research, we use the ingredients information from the Spoonacular API to calculate the cost of each missing ingredient in each recipe given by CAPRecipes. The API request endpoint for determining the price of ingredients as per their quantity is as follows [Mar18]:

```
POST https://spoonacular-recipe-food-nutrition-v1.p.mashape.com
↪ /recipes/visualizePriceEstimator
```

When this API is initiated, the ingredient information, including name, quantity and unit, and the API key are sent to this service. For example, the code snippet in Listing 3.3 depicts the request body sent to this service to determine the price of 0.13 cup of tomatoes.

```
{
  "defaultCss": True,
  "ingredientList": "0.13 cup tomatoes",
  "mode": 1,
```

```
  "servings": 1,
  "showBacklink": True
}
```

Listing 3.3: JSON code snippet of request body sent to the Spoonacular Grocery API

After receiving the HTTP *200* response along with the HTML output, the price of an ingredient is extracted with the help of JSON key *dataPoints*. For example, 0.13 cup tomatoes cost 6 cents as shown in Listing 3.4, which is extracted from HTML response.

```
{
  "image": "tomato.jpg",
  "amount": "1/8 cup",
  "price": "6 cents",
  "y": "6",
  "indexLabel": "tomatoes"
}
```

Listing 3.4: Simplified version of the Spoonacular Grocery API response

The pseudo code below describes the calculation of price for each missing ingredient present within a recipe, and the total missing ingredients cost of each recipe from the recommendation list. The full source code is in Appendix A.1.

Algorithm 2 Calculation of missing ingredients cost of recipes

for each recipe in recommended recipes list from CAPRecipes **do**
  Set $total\_missing\_ingredient\_cost = 0$
  for each missing ingredient in recipe **do**
    Create *ingredient_information* string using name, amount and unit from recipe JSON string
    Query the **Visualize Price Breakdown API request** using *ingredient_information*
    Extract *price* from API response
    $total\_missing\_ingredient\_cost = total\_missing\_ingredient\_cost + price$
    Add *price* attribute to each missing ingredient JSON string
  end for

Add *total_missing_ingredient_cost* attribute to each recipe JSON string
end for
return recipe list with missing ingredient costs information

In addition to the use of these API services in building SmartGrocer, we also developed two separate algorithms to achieve other functionalities of SmartGrocer. These algorithms are illustrated in the next sections.

3.3 POS Tagging Mean Probability Algorithm

During the process of automatically creating the personalized grocery list, SmartGrocer interacts with several services such as CAPRecipes as a third-party recipe recommendation system, the Foodpairing API for generating the ingredient pairings, and the Spoonacular API for calculating the cost of recipes. While making these interactions, the problem that arises is the lack of structure in the names of ingredients used in recipes and in the various API responses. For example, an ingredient *juice of lime* retrieved from a recipe might not be available with the same name in the grocery database. Thus, we present an information retrieval method, which is a Part-Of-Speech (POS) Tagging Mean Probability method [ES15] that enables us to perform a search and find the perfect or the most relevant match of each recipe ingredient or ingredients from the Foodpairing service in the Grocery API ingredient database.

POS tagging, also known as word classes or lexical categories [Nlt18], is useful for simple analysis of the distribution of words in text that gives us a simple context in which to present them.

In our research, we use the POS Tagging Mean Probability algorithm for the following scenarios:

1. To replace an ingredient name in each recipe with the grocery ingredient name.

2. To extract IDs of queried ingredients from the response of the Foodpairing *Search Ingredients* API request.

3. To replace ingredient names obtained from the Foodpairing *Ingredients Pairing* API request with the grocery ingredient names.

For example, to replace any ingredient name in a recipe with the grocery store ingredient name, we first use the POS tagging service to identify nouns, verbs, and adjectives of a recipe's ingredient. Nouns have most of the ingredient information, adjectives define the ingredient in the most specific form, such as *fresh* or *frozen*, and verbs in most cases state the preparation method, for example, *cooked*, *canned* and so on [ES15]. Second, we query an API request to the grocery database using the provided nouns. Third, we again apply POS tagging on each ingredient in the grocery API response. Fourth, we compute the probability in the form of similarity coefficient of nouns, adjectives, and verbs of an ingredient name and each of the ingredient names of the grocery API response respectively. In the end, we compute the mean probability, and assigned it to matching pairs. The matching pair with the highest mean probability is considered to be the most relevant match. The following flow diagram explains the POS Tagging Mean Probability algorithm.

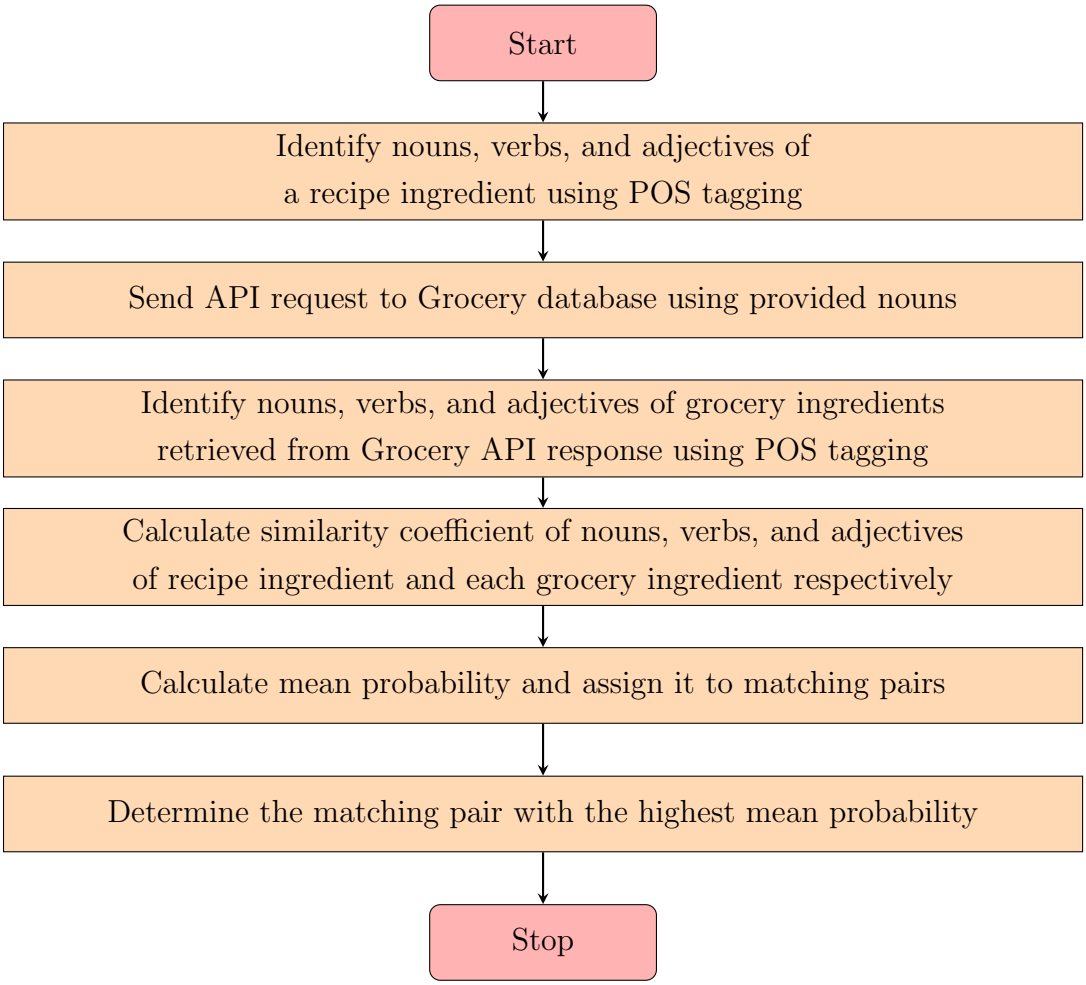


Table 3.1, shows the most relevant or perfectly matched ingredient name from the grocery database after applying the POS Tagging Mean Probability algorithm on some of the recipe ingredients.

Table 3.1: Most relevant match of recipe ingredients with grocery ingredients

Ingredient name from recipe	**Most relevant ingredient name from grocery database**
Brown rice	Dynasty Brown Rice
Green bell pepper	Green Bell Peppers
Juice of lime	Safeway Lime Juice
Roma tomato	Red Roma Tomatoes
Whole wheat tortillas	La Tortilla Factory Fat Free Whole Wheat Tortillas

Currently, the other names of ingredients, for example, scallions is also called as green onions or spring onions, are not appropriately handled, and these ingredients are considered to be as missing ingredients by SmartGrocer. The pseudo code below outlines the POS Tagging Mean Probability algorithm for searching the most relevant or perfect match for a recipe ingredient in the grocery database. The source code of this algorithm is in Appendix A.3.

Algorithm 3 POS Tagging Mean Probability algorithm

1: **for** each ingredient name in recipe **do**
2: Set $matching\_pairs = list()$
3: Set $P(N) = 0$ ▷ Set initial noun(N) probability to zero
4: Set $P(V) = 0$ ▷ Set initial verb(V) probability to zero
5: Set $P(A) = 0$ ▷ Set initial adjective(A) probability to zero
6: Set $counter = 0$
7: Set $matching\_threshold = 0.7$ ▷ Set the minimum probability for matching nouns, verbs, and adjectives to 0.7
8: Set $min\_mean\_probability = 0.7$ ▷ Set the minimum mean probability to 0.7
9: Extract the sets of nouns R_N, verbs R_V, and adjectives R_A of a recipe ingredient using POS tagging service
10: Query the **Grocery API** using the set of provided nouns R_N

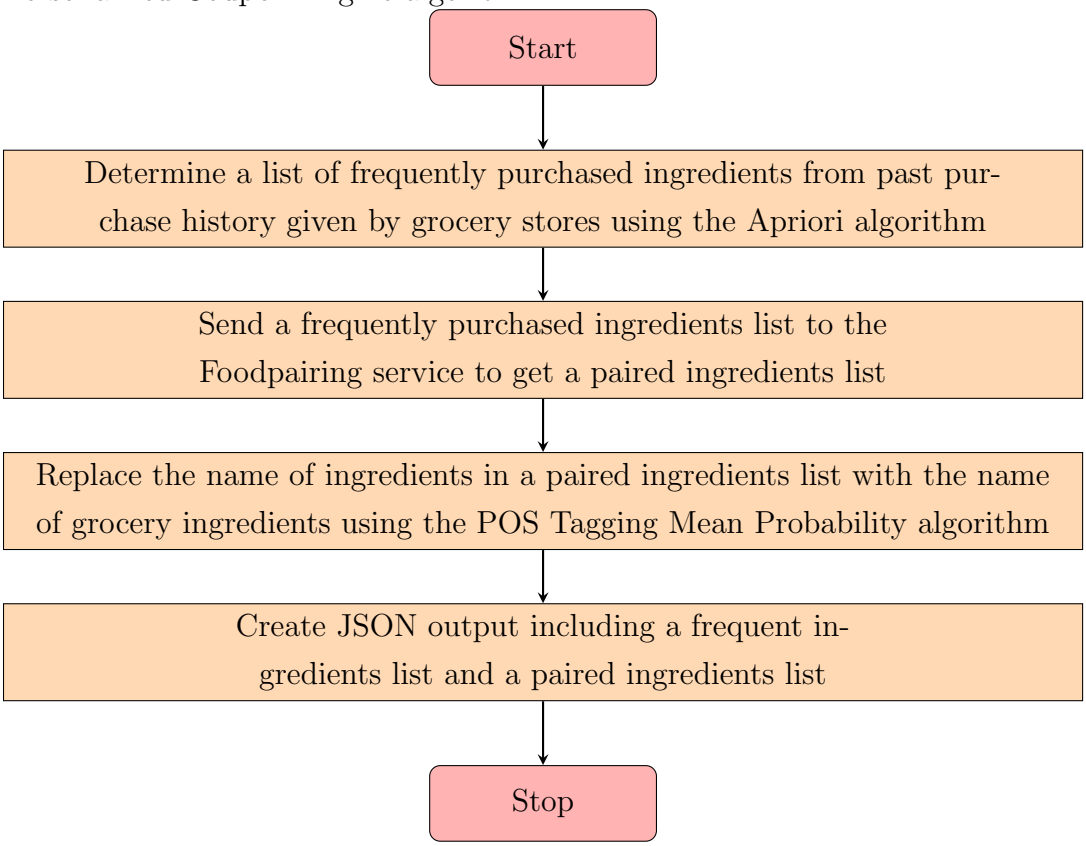
11: **for** each ingredient name from the grocery API response **do**
12: Extract the sets of nouns *G_N*, verbs *G_V*, and adjectives *G_A* of a grocery ingredient using POS tagging service
13: Calculate the probability as the string similarity coefficient between *R_N*, *R_V*, *R_A* and *G_N*, *G_V*, *G_A* respectively
14: **if** *noun_probability* > *matching_threshold* **then**
15: *P(N)* += *noun_probability*
16: **end if**
17: **if** *verb_probability* > *matching_threshold* **then**
18: *P(V)* += *verb_probability*
19: **end if**
20: **if** *adjective_probability* > *matching_threshold* **then**
21: *P(A)* += *adjective_probability*
22: **end if**
23: $P(X) = (P(N) + P(V) + P(A))/3$ ▷ Calculate mean probability
24: **if** *P(X)* > *min_mean_probability* **then**
25: *matching_pairs[counter]* = *P(X)*
26: *counter*+ = 1
27: **end if**
28: **end for**
29: **return** the most relevant match, *max(matching pairs)*
30: **end for**

3.4 Personalized Coupon Engine Algorithm

SmartGrocer provides personalized coupons to users with the aim of increasing saving and reducing food waste in stores. These coupons are dynamically generated and are specific to each user. We use the Apriori algorithm of Association Rules (c.f., Chapter 2), the Foodpairing service, and the POS Tagging Mean Probability algorithm to generate personalized coupons.

To create personalized coupons, first a list of frequently purchased ingredients are derived from the user's past purchase histories given by grocery stores using the Apriori algorithm. Then, we send a list of frequently purchased ingredients to the Foodpairing service to get a paired ingredients list. The names of the ingredients in

a paired ingredients list is then replaced with the grocery ingredient names using the POS Tagging Mean Probability algorithm. Now, we have two lists of ingredients: one is a list of frequently purchased ingredients, and another is a list of paired ingredients, which are arranged in JSON format. These two lists of ingredients are then compared to the list of soon-to-expire and promotional ingredients, including their names and discount rates, given by the Grocery API. The following flow diagram describes the Personalized Coupon Engine algorithm.



In the end, we have a JSON output that includes frequently purchased ingredients and paired ingredients list along with their discount rates as shown in Listing 3.5.

```
{
"Data": [
  {
    "FrequentIngredients": [
      {"name":"bell pepper", "discount":"15%"},
      {"name":"apple", "discount":"15%"},
      {"name":"croissant", "discount":"20%"},
      {"name":"strawberry", "discount":"15%"},
      {"name":"carrot", "discount":"15%"},
      {"name":"orange", "discount":"15%"},
```

```
        {"name":"egg", "discount":"20%"}
      ],
      "PairedIngredients": [
        {"name":"Spinach", "discount":"15%"},
        {"name":"Gruyere", "discount":"10%"},
        {"name":"Lychee", "discount":"20%"},
        {"name":"Tarragon", "discount":"10%"},
        {"name":"Sweet cherry", "discount":"15%"}
      ]
    }
  ]
}
```

Listing 3.5: Simplified view of the response from Personalized Coupon Engine algorithm

The pseudo-code below explains the Personalized Coupon Engine algorithm for creating personalized coupons. The complete source code is available in Appendix A.2.

Algorithm 4 Personalized Coupon Engine Algorithm

Set $k = 1$
Set *Frequent_Ingredients* = *list()*
Set *Paired_Ingredients* = *list()*
Set *Final_Frequent_Ingredients* = *list()*
Set *Final_Paired_Ingredients* = *list()*
Set *min_support* = 0.15 ▷ Setting of *min_support* value between 0.1 and 0.2
Set F_k = frequent-k itemsets
$k+ = 1$
for each k until F_{k-1} is not *null* **do**
    Generate Candidate set, C_k from F_{k-1}
    for each transaction, t in database, D **do**
        Extract transaction sets which contained t, C_t = *subset*(C_k, t)
        for each candidate, C in C_t **do**
            C_count $+ = 1$
        end for
    end for

17: **if** *C_count*> *min_support* **then**
18: F_k = candidates in C_k
19: **end if**
20: $k+ = 1$
21: **end for**
22: *Frequent_ingredients* = $\cup_k F_k$
23: Query the Foodpairing service using *Frequent_ingredients*, *Paired_ingredients* = *FoodPairing(Frequent_ingredients)*
24: Match the *Paired_ingredients* list with grocery ingredients database, *Paired_ingredients* = *pos_tag(Paired_ingredients,Grocery_ingredients_list)*
25: Generate *Final_Frequent_Ingredients* with discount_rates = *Frequent_ingredients* available in soon to-expire and promotional ingredients list from Grocery API
26: Generate *Final_Paired_Ingredients* with discount_rates = *Paired_ingredients* available in soon to-expire and promotional ingredients list from Grocery API
27: **return** {*Final_Frequent_Ingredients,Final_Paired_ingredients*}

3.5 Summary

This chapter presented algorithms that call different REST API services, which are the Spoonacular API that is used to determine the cost of recipes and the Foodpairing service that is used to generate the ingredients list, which can be paired well with frequently purchased ingredients. This chapter also explained two algorithms created to achieve SmartGrocer's goals, which are to save users money and promote the reduction of food waste in stores. These algorithms are a POS Tagging Mean Probability algorithm and a Personalized Coupon Engine algorithm. The POS Tagging Mean Probability algorithm is used to match and replace ingredient names retrieved from different sources, with the most relevant match obtained from the grocery database. The Personalized Coupon Engine algorithm is used to provide personalized coupons to users based on their past purchase histories, Foodpairing results, and the grocery store's information on soon-to-expire and promotional ingredients.

Chapter 4

System Architectural Design and Implementation

This chapter focuses on the architectural system design of SmartGrocer, which consists of several critical components. It first describes the functionality of each component of our system. Then the workflow of the system is explained with the most significant features being implemented through a web application.

4.1 SmartGrocer

4.1.1 Overview

SmartGrocer is designed with the aim to provide benefits to both users and grocery store businesses by optimizing a user's grocery budget and promoting the reduction of food waste in stores. As stated in Chapter 1, users face various challenges before they plan their grocery visit. They have to spend a considerable amount of time and energy creating a grocery list that might involve some decisive steps. These steps are planning of a daily meal routine according to their taste, health, and budget, and looking out for grocery coupons from flyers or third-party grocery deal recommendation applications manually that users can use in their upcoming grocery trip. Also, the current grocery applications that are available on the market provide general grocery coupons only. These applications are not using store product inventory and user context to provide personalized discount coupons.

In light of this, we developed SmartGrocer as a proof of concept. It is a Context-Aware Personalized Grocery System that assists users in planning their meals while

considering their budget, applies personalized coupons for missing ingredients on the fly, and creates a personalized grocery list automatically after selecting recipes for preparation. It leverages user and grocery store context to generate personalized grocery coupons which result in saving money and also promotes the better management of the store inventory. The high-level architecture of the SmartGrocer is shown in Figure 4.1.

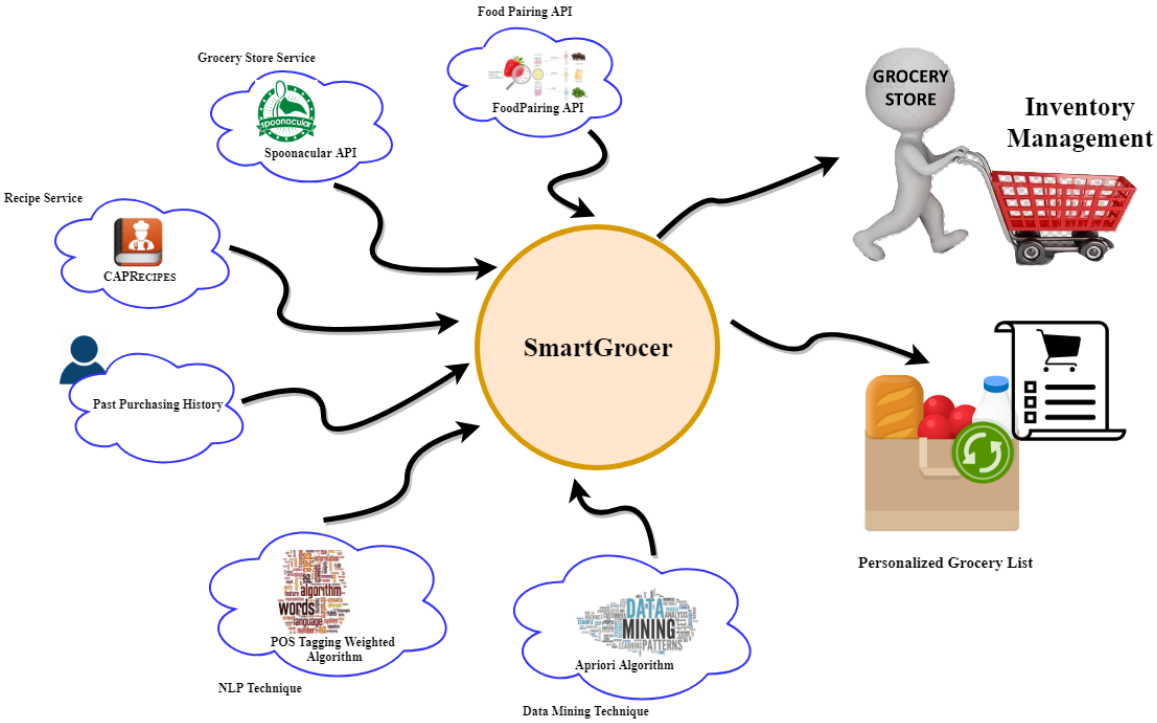


Figure 4.1: High-level architecture of SmartGrocer

The system is split into six components, namely *Recipe Engine*, *Grocery Context*, *Users Context*, *Personal Coupon Engine*, *Recipe Cost Optimization* and *Personalized Grocery List Generator* (depicted in Figure 4.2). Each component plays a significant role in achieving the system goals and is elaborated in the following sections.

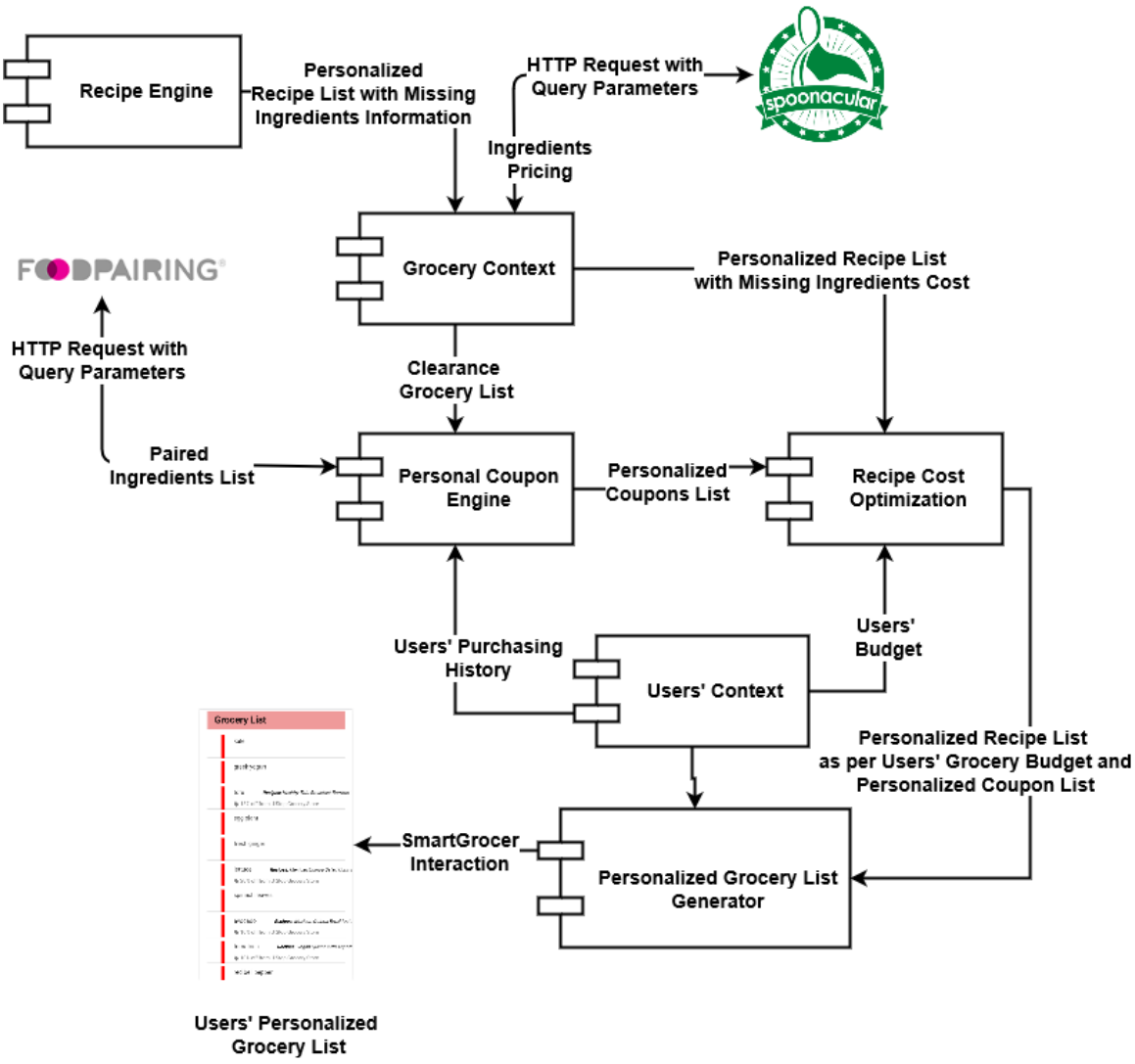


Figure 4.2: Component diagram of SmartGrocer

4.1.2 Recipe Engine Component

On application start, SmartGrocer receives a list of recipes from a third-party recipe recommendation system, which must include information related to at least used and missing ingredients for recipes. For our research, we integrated the Context-Aware Personalized Recipe recommendation system (CAPRecipes) as a third-party recipe recommendation system in the Recipe Engine component. CAPRecipes provides recipe recommendations by recognizing user health goals and restrictions, availability of ingredients in their kitchen, and dynamically changing taste preferences. This system eliminates the manual browsing of recipes that fit a user's health and taste.

It also maximizes the number of available ingredients while recommending recipes to users. The output of the Recipe Engine component is a list of personalized recipes organized in JSON format that includes but is not limited to recipe name, nutrition details (i.e., calories, carbs, proteins, and fat), preparation time and missing ingredients information [Jai18]. The Listing 4.1 shows attributes of one recipe from the Recipe Engine component.


```
{  "data":
 [
  {
    "carbs": "90g",
    "title": "Vegetarian Grilled Mexican Stuffed Bell Peppers",
    "cuisines": [ "mexican" ],
    "calories": 612,
    "fat": "12g",
    "readyInMinutes": 30,
    "usedIngredients": [
      {
        "aisle": "Canned and Jarred",
        "name": "beans",
        "amount": 0.5,
        "id": 16069,
        "unit": "cups"
      },
      {
        "aisle": "Produce",
        "name": "bell peppers",
        "amount": 1,
        "id": 10211821,
        "unit": ""
      },
      {
        "aisle": "Produce",
        "name": "fresh cilantro",
        "amount": 0.75,
        "id": 11165,
        "unit": "tablespoons"
      }
    ],
    "missingIngredients": [
      {
        "aisle": "Pasta and Rice",
```

```
            "name": "salsa",
            "amount": 0.08,
            "id": 6164,
            "unit": "cup"
          },
          {
            "aisle": "Pasta and Rice",
            "name": "brown rice",
            "amount": 0.5,
            "id": 20041,
            "unit": "cups"
          },
          {
            "aisle": "Cheese",
            "name": "sharp cheddar cheese",
           "amount": 1,
            "id": 1031009,
            "unit": "servings"
          }
        ],
        "protein": "36g",
        "id": 584043
      }
    ]
}
```


Listing 4.1: Simplified view of JSON response from Recipe Engine component

4.1.3 Grocery Context Component

Grocery stores have various forms of data that we can exploit to provide different useful services to users, which might satisfy them and also promote the store businesses. Some of these store data such as ingredients pricing, user transaction histories, and grocery stock information, and are used in our research to accomplish the goals of SmartGrocer. The Grocery Context component includes data mainly related to grocery items, such as ingredients pricing and the grocery stock information of soon-to-expire and promotional ingredients, to calculate the cost of missing ingredients and to generate personalized coupons respectively. As we do not have permissions to access any particular grocery store dataset, we used the grocery part of the Spoonacular service as a ingredients pricing database for our research to compute the cost of

missing ingredients for each recipe. Due to lack of access, we also created a grocery inventory list with the list of ingredients which are going to expire soon, or are new and need to be put on promotion with their discount information for SmartGrocer realization. This list is named as Clearance Grocery List and is used when recommending personalized coupons to users. The information in this list is arranged in JSON as shown in Listing 4.2:

```
{ "Data":
 [
  {
    "IngredientsList": [
      {"name":"black beans", "discount":"15%", "expiryDate":
"14/05/2018"},
      {"name":"avocado", "discount":"10%", "expiryDate":
"18/04/2018"},
      {"name":"potato", "discount":"10%", "expiryDate": "18/04/2018"},
      {"name":"lettuce", "discount":"20%", "expiryDate":
"17/04/2018"},
      {"name":"tomatoes", "discount":"10%", "expiryDate":
"17/04/2018"},
      {"name":"strawberry", "discount":"15%", "expiryDate":
"17/04/2018"},
      {"name":"brown rice", "discount":"20%", "expiryDate":
"19/05/2018"},
      {"name":"lindt chocolate", "discount":"15%"},
      {"name":"okra", "discount":"10%", "expiryDate": "18/05/2018"},
      {"name":"red apples", "discount":"15%", "expiryDate":
"18/05/2018"},
      {"name":"oranges", "discount":"10%", "expiryDate":
"18/05/2018"},
      {"name":"Maggie tamarind sauce", "discount":"15%", "expiryDate":
 "28/05/2018"}

    ]
  }
 ]
}
```

Listing 4.2: Simplified view of Clearance Grocery List

The personalized recipes list from the Recipe Engine Component is given as an input to this component which produces another personalized recipes list. This list consists

of additional attributes other than the primary attributes, which are: *price* (price of each missing ingredient) and *missingIngredientsCost* (total cost of missing ingredients of a recipe in dollars) for each recipe. The algorithm to determine the price of the missing ingredients and the total missing ingredients cost of a recipe is described in Chapter 3, Section 3.2. Another output from this component is a Clearance Grocery List. The Listing 4.3 is a JSON snippet that presents the simplified version of a recipe resulted from the Grocery Engine component, which includes the *price* for one of the missing ingredients and *missingIngredientsCost* information.


```
{
  "data":
  [
    {
      "carbs": "90g",
      "title": "Vegetarian Grilled Mexican Stuffed Bell Peppers",
      "cuisines": [
        "mexican"
      ],
      "calories": 612,
      "fat": "12g",
      "readyInMinutes": 30,
      "missedIngredientsCost": 0.5,
      "missedIngredients": [
        {
          "aisle": "Pasta and Rice",
          "name": "salsa",
          "amount": 0.08,
          "price": "$0.08",
          "id": 6164,
          "unit": "cup"
        }
      ],
      "protein": "36g",
      "id": 584043
    }
  ]
}
```


Listing 4.3: Simplified view of JSON response from Grocery Engine component

4.1.4 Users Context Component

User related data from grocery stores such as their purchase history is used as a part of the Users Context component. It also includes personal information, which is their meal budget. The meal budget is manually gathered from the user through our web interface. The sample of purchasing history of the user is created for the realization of our system and is stored in the tabular manner as shown in Table 4.1:

Table 4.1: Simplified view of purchasing history of a user

Transaction ID	**Items Purchased**
1	apple, egg, bell pepper, carrot, croissant
2	egg, black tea, strawberry, orange, bell pepper
3	carrot, croissant, black tea, orange
4	bell pepper, egg, strawberry, orange, apple
5	apple, egg, orange, bell pepper, black tea
6	orange, croissant, orange, strawberry
7	carrot, strawberry, bell pepper
8	croissant, black tea
9	apple, black tea, egg, strawberry
10	carrot, orange, croissant

The Users Context component is dynamically updated everytime a user completes their grocery purchases from stores.

4.1.5 Personalized Coupon Engine Component

The Personalized Coupon Engine (PCE) component is the core component of Smart-Grocer. This component utilizes the Clearance Grocery List provided by the Grocery Context component and the purchasing histories of users from the Users Context component to recommend personalized coupons to users. We also use the Foodpairing service, which provides additional ingredients that can be paired well with the frequently purchased ingredients of users. The sample output of the PCE component after applying the PCE algorithm as explained in Chapter 3 Section 3.4 is a list of frequent and paired ingredients with their discount information as shown in Listing 4.4:

```
{ "Data":
 [
    {
      "FrequentIngredients": [
        {"name":"black beans", "discount":"15%"},
        {"name":"avocado", "discount":"10%"},
        {"name":"potato", "discount":"10%"},
        {"name":"lettuce", "discount":"10%"},
        {"name":"tomatoes", "discount":"10%"},
        {"name":"strawberry", "discount":"15%"}
      ],
      "PairedIngredients": [
        {"name":"brown rice", "discount":"20%"},
        {"name":"lindt's choclates", "discount":"15%"},
        {"name":"cream cheese", "discount":"10%"},
        {"name":"tortillas", "discount":"15%"},
        {"name":"lemon", "discount":"10%"},
        {"name":"carrots", "discount":"10%"},
        {"name":"spring onions", "discount":"10%"}
      ]
    }
  ]
}
```


Listing 4.4: Simplified view of JSON response from Personalized Coupon Engine component

4.1.6 Recipe Cost Optimization Component

The Recipe Cost Optimization component applies personalized coupons obtained from the PCE component to the missing ingredients of the personalized recipes derived from the Grocery Context component. This component reduces the cost of each recipe, and the reduced cost is displayed in the JSON of personalized recipes as an extra attribute, named as *missingIngredientsCostAfterDiscount* (in dollars) for each recipe as illustrated in Listing 4.5. The output of this component is the final personalized recipe list, which includes the reduced cost of recipes that would help users in selecting recipes as per their budget.


```
{
  "data":
    [
```

```
    {
      "carbs": "90g",
      "title": "Vegetarian Grilled Mexican Stuffed Bell Peppers",
      "cuisines": [
        "mexican"
      ],
      "calories": 612,
      "fat": "12g",
      "readyInMinutes": 30,
      "missedIngredientsCost": 0.5,
      "missingIngredientsCostAfterDiscount": 0.38,
      "missedIngredients": [
        {
          "aisle": "Pasta and Rice",
          "name": "salsa",
          "amount": 0.08,
          "price": "$0.08",
          "id": 6164,
          "unit": "cup"
        }
      ],
      "protein": "36g",
      "id": 584043
    }
  ]
}
```


Listing 4.5: Simplified view of JSON response from Recipe Cost Optimization component

4.1.7 Personalized Grocery List Generator Component

This component generates personalized grocery list automatically when the user selects a recipe to prepare, or after creating the meal plan for few days/weeks. The list includes the list of ingredients including name, discount coupon, and the recipes which are using these ingredients. Also, this component recommends coupons for those ingredients that are not used in the selected recipes, but match with the user's frequent or paired ingredients coupons list obtained from the PCE component.

4.2 SmartGrocer Web Interface

To demonstrate the capabilities of SmartGrocer implemented as a part of this thesis, a web application is created to enrich the user experience. The application recommends recipes to users based on their budget after applying personalized coupons to the missing ingredients in their recipes, and assists in creating a meal plan and a personalized grocery list automatically. The following sections describe the significant features of SmartGrocer.

4.2.1 Recipe Recommendation Page

The main page of the SmartGrocer application shows the recipe recommendations with their original and reduced cost after using the personalized coupons. These recipes are arranged according to the user's budget. Recipes whose final missing ingredients cost after discount remains within the user's budget are displayed with the *green* bottom line, and the recipes which are not, are displayed with the *red* bottom line. For example, a user has a dollar budget per day for the main course of lunch, then the recipes with missing ingredients cost less than a dollar after discounts are displayed with *green* bottom line, and the recipes with higher reduced missing ingredients costs are displayed with the *red* bottom line, depicted in Figure 4.3.

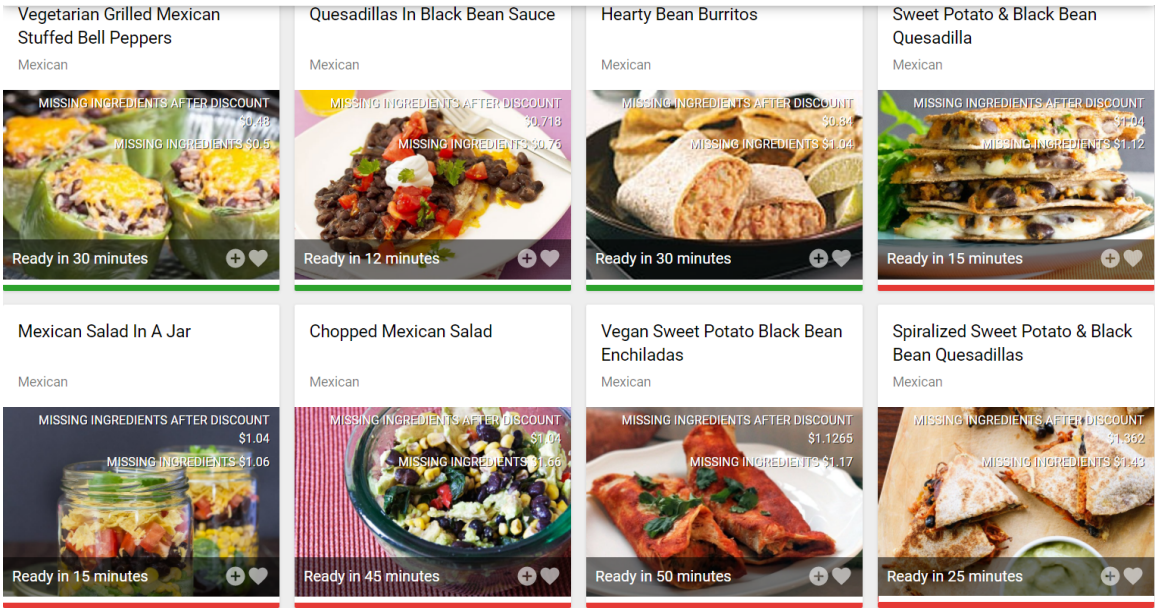


Figure 4.3: Recipe Recommendation page

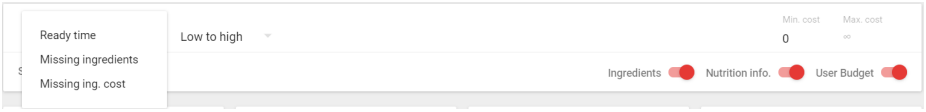
These recipes can also be arranged using the following options from the dropdown menu as shown in Figure 4.4a and 4.4b. It organizes the recipes list either in ascending or descending order by selecting *Low to High* or *High to Low* option respectively.

1. **By Missing Ingredients Cost (after discount)** - This option orders recipes based on the cost of missing ingredients of each recipe after applying the personalized coupons.

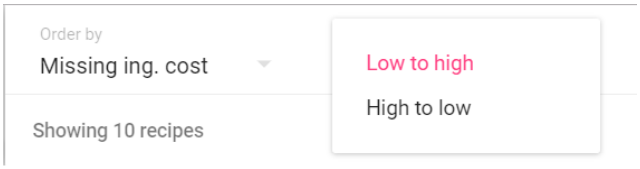
2. **By Number of Missing Ingredients** - This option orders recipes based on the number of missing ingredients available in each recipe.

3. **By Ready Time** - This option orders recipes based on their preparation time.

Furthermore, the recipe recommendations while considering ingredients and user context such as ingredient details and nutrition information of each recipe and budget can be turned off at any time using the toggle functionality, shown in Figure 4.4a to get general recipe recommendations.



(a) Options of recipe filtering



(b) Sorting order of recipes

Figure 4.4: Recipes filtering section

4.2.2 Recipe Detail Page

On selecting any recipe from the Recipe Recommendation page, a Recipe Detail page is displayed from where a user can find more details of a recipe, such as steps for preparation, equipment needed, and ingredients used in each step. The user can also add all the missing ingredients directly to her grocery list from this page, depicted in Figure 4.5.

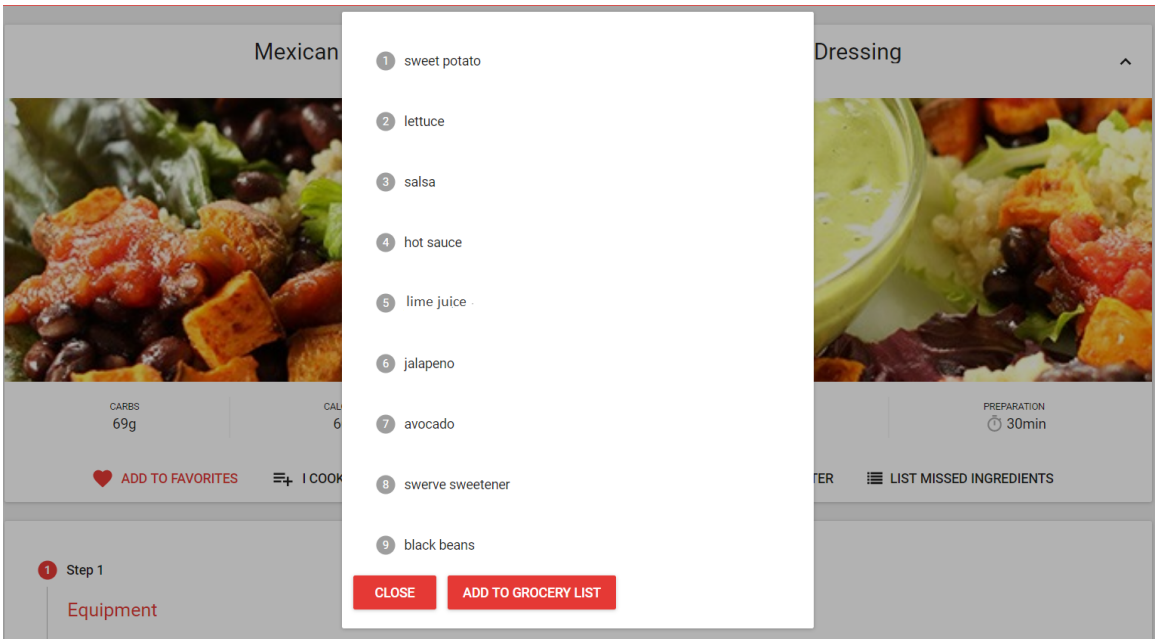


Figure 4.5: Recipe Detail page

4.2.3 User Controls Menu

The user can visit different pages with the help of a User Controls Menu, which is shown as a dropdown view in Figure 4.6. This menu includes reference to *My profile*, *Favorite Recipes*, *Wishlist*, *Meal Plan* and *Grocery List* page.

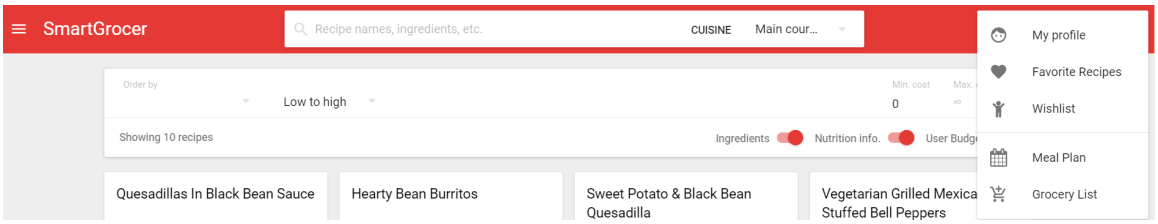


Figure 4.6: User Controls Menu page

4.2.4 My Profile Page

The per-day meal budget for breakfast, lunch, and dinner are gathered manually from the user through My Profile page under *User Daily Budget* section, shown in Figure 4.7. This page collects other information, such as allergies, diet, nutrition information (i.e., daily requirement of carbs, proteins, and fat) and taste preferences. The user can update these details at any time on this page.

Figure 4.7: My Profile page [Jai18]

4.2.5 Wishlist Page

This page consists of all those items children need from the store. The items from the Wishlist page move to the Grocery List page after parental confirmation. This feature is useful as it makes adults aware of their children's needs and also provides a chance to help them make healthy choices. Figure 4.8 shows the Wishlist page. *MOVE TO GROCERY LIST* option is disabled in the children account.

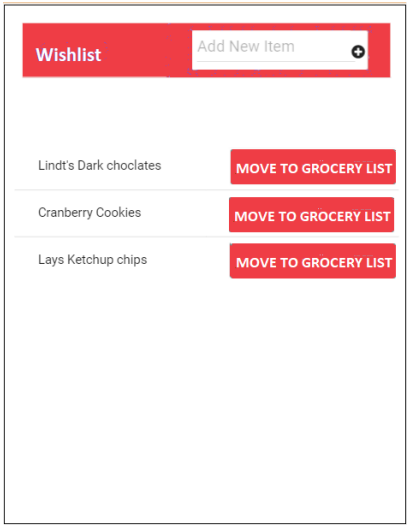


Figure 4.8: Wishlist page

4.2.6 Meal Plan Page

The user can create a meal plan for few weeks or less from the Meal Plan page. Initially, the recipes for breakfast, lunch, and dinner are automatically selected when the user sets the *Start* and *End* date. The user can modify these recipes with other recommended recipes or move these recipes between days or meals. The Meal Plan page also displays the summary of each day which includes the cost of missing ingredients before and after discount, savings and nutrition details. The summary of a day keeps track of daily nutrition intake as well as savings. Figure 4.9 shows the meal plan for six days. The grocery list is automatically created, once the user finalizes the meal plan.

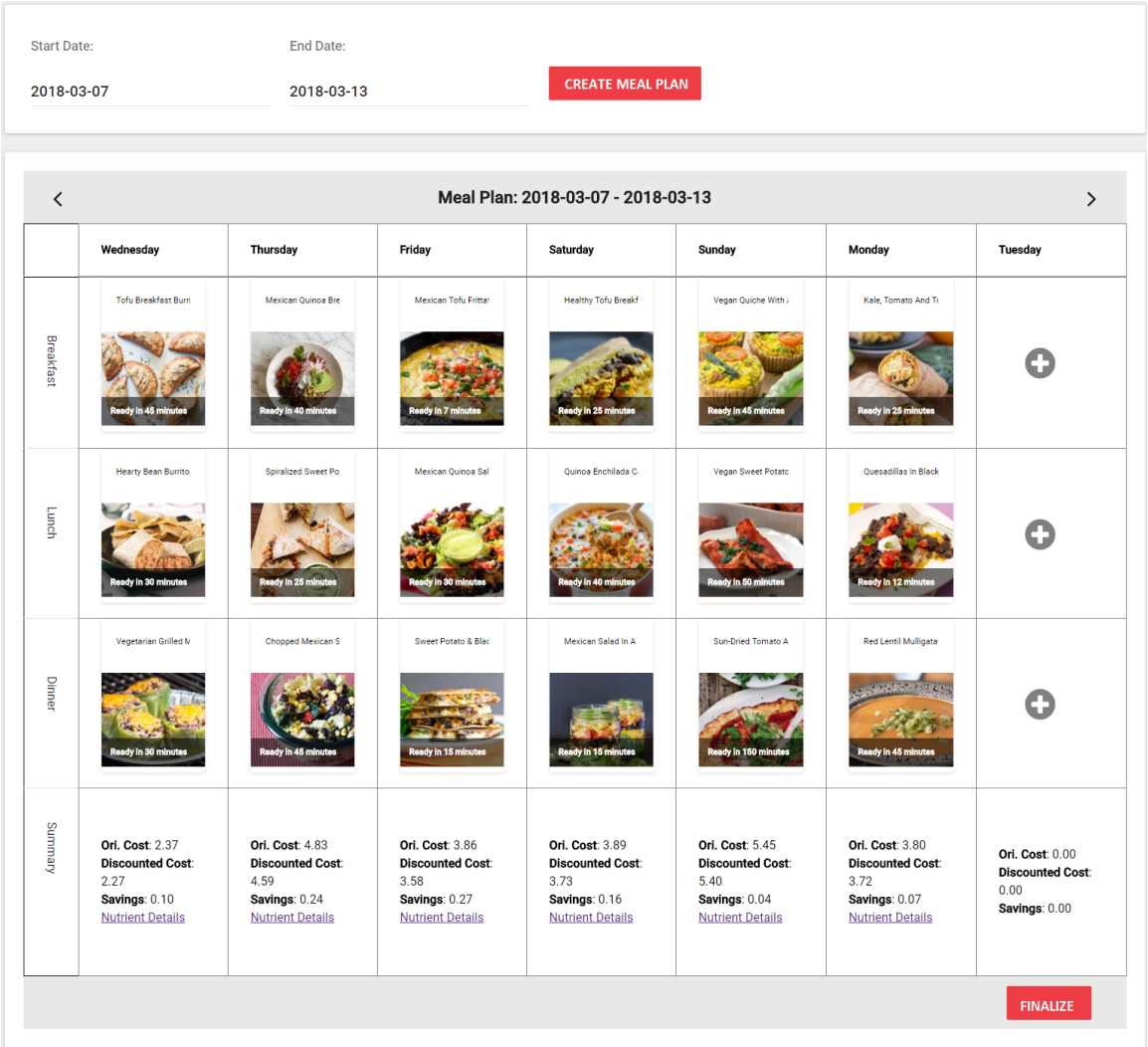
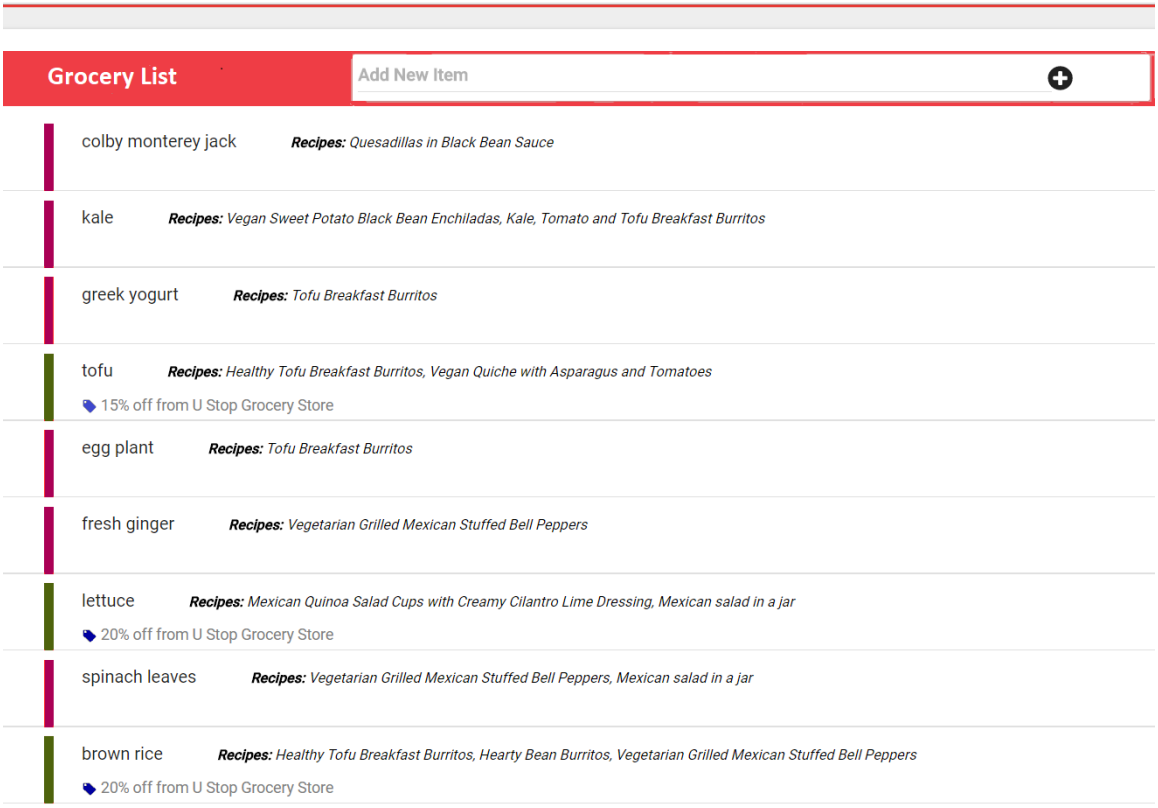


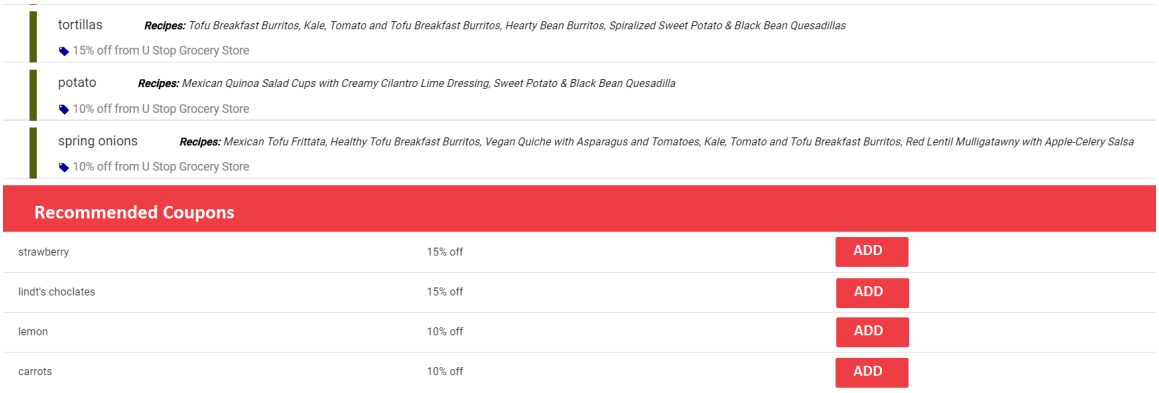
Figure 4.9: Meal Plan page

4.2.7 Grocery List Page

This page consists of the list of ingredients that the user would like to purchase. The ingredients can be the missing ingredients of the selected recipes or manually added ingredients. The Grocery List page displays the ingredients name, personalized discount coupons and the list of recipes which use these ingredients. This page also includes the *Recommended Coupons* section for those ingredients, which are not on the Grocery List but match the frequently purchased and paired ingredients coupons list obtained from the PCE component. Figure 4.10 shows the *Grocery List* and the *Recommended Coupons* sections of the Grocery List page.



(a) Personalized grocery list



(b) Recommended Coupons section

Figure 4.10: Grocery List page

4.3 Summary

This chapter presented key architectural and functional features of SmartGrocer. SmartGrocer is a Context-Aware Personalized Grocery system that helps users with grocery shopping according to their budget. The system automates the repetitive

steps involved, such as selection of recipes, browsing of grocery coupons, and the creation of personalized a grocery list, by exploiting user and store contextual information through the SmartGrocer web interface.

Chapter 5

Evaluation, Analysis and Comparisons

In this chapter we evaluate the performance and usefulness of SmartGrocer from both the user and store perspective. We present realistic scenarios to show different functionalities of the system. The performance of the system is measured based on *Efficiency*, *Effectiveness* and *User Experience*. We also compare recipe recommendations from SmartGrocer with CAPRecipes through user feedback. However, we did not perform formal user studies.

5.1 Evaluation of SmartGrocer

The implementation of the prototype application SmartGrocer is considered to be the part of evaluation. Moreover, the usefulness and the acceptance of SmartGrocer are evaluated using two experiments, which were performed separately with volunteers on 14 April, 2018. The personal information of volunteers involved in the experiments has been anonymized due to privacy issues. In Experiment 1, we illustrate the scenario of a young male who wants recipes according to his taste preferences and health restrictions while considering his limited budget, storage space, and lack of time to plan grocery shopping. Experiment 2 describes the challenges faced by a senior woman in creating a grocery list according to her meal plan. As our application is in its prototype stage, we built our own grocery store for evaluation purposes, known as *U Stop Grocery Store* which has limited ingredients information.

5.2 U Stop Grocery Store

The store database includes ingredient information — *name*, *quantity*, *unit*, *expiry date*, *price*, and *promotional offer*, if any. We created a store's Clearance Grocery List from an ingredients database in JSON format, which includes those ingredients that are soon-to-expire or have a promotional offer on them. The JSON string consists of name, expiry date (optional), and store discount given by the grocery store. Also, the volunteers of our experiments have shared receipts of their grocery purchases of the last two years with our store. The JSON string shown in Listing 5.1 is a snapshot of the store's Clearance Grocery List, which was created on 14 April, 2018.


```
{
"Data":
 [
  {
  "IngredientsCoupons":
   [
  {"name":"black beans", "discount":"15%", "date": "14/05/2018"},
  {"name":"avocado", "discount":"10%", "date": "18/04/2018"},
  {"name":"potato", "discount":"10%", "date": "18/04/2018"},
  {"name":"lettuce", "discount":"20%", "date": "17/04/2018"},
  {"name":"tomatoes", "discount":"10%", "date": "17/04/2018"},
  {"name":"strawberry", "discount":"15%", "date": "17/04/2018"},
  {"name":"brown rice", "discount":"20%", "date": "19/05/2018"},
  {"name":"lindt chocolate", "discount":"15%"},
  {"name":"okra", "discount":"10%", "date": "18/05/2018"},
  {"name":"red apples", "discount":"15%", "date": "18/05/2018"},
  {"name":"oranges", "discount":"10%", "date": "18/05/2018"},
  {"name":"Maggie tamarind sauce", "discount":"15%", "date":
   "28/05/2018"}
  {"name":"Whole wheat bread", "discount":"15%", "date": "18/05/2018"}
  {"name":"cream cheese", "discount":"10%", "date": "25/04/2018"},
  {"name":"tortillas", "discount":"15%", "date": "15/05/2018"},
  {"name":"lemon", "discount":"10%", "date": "17/04/2018"},
  {"name":"carrots", "discount":"10%", "date": "18/04/2018"},
  {"name":"spring onions", "discount":"10%", "date": "18/04/2018"},
  {"name":"Canned Chickpeas", "discount":"10%", "date": "30/04/2018"},
  {"name":"Cashews", "discount":"20%"},
  {"name":"Cauliflower", "discount":"15%", "date": "17/04/2018"},
  {"name":"Mushrooms", "discount":"15%", "date": "18/04/2018"},
  {"name":"Kale", "discount":"15%", "date": "17/04/2018"},
```

```
  {"name":"Ginger", "discount":"10%", "date": "18/04/2018"},
  {"name":"Garlic", "discount":"10%", "date": "18/04/2018"},
  {"name":"Tofu", "discount":"15%", "date": "30/04/2018"},
  {"name":"Broccoli", "discount":"15%", "date": "18/04/2018"},
  {"name":"Salmon", "discount":"15%", "date": "18/04/2018"},
  {"name":"Eggs", "discount":"10%", "date": "17/04/2018"},
  {"name":"Peas", "discount":10, "date": "18/04/2018"},
  {"name":"Shrimp", "discount":"15%", "date": "18/04/2018"},
  {"name":"Coconut Milk", "discount":"15%", "date": "19/04/2018"},
  {"name":"Pork Loin", "discount":"15%", "date": "18/04/2018"},
  {"name":"Cabbage", "discount":"15%", "date": "18/04/2018"},
  {"name":"Gruyere", "discount":"15%", "date": "18/04/2018"},
  {"name":"Cardamom", "discount":"20%", "date": "28/06/2018"},
  {"name":"Bacon", "discount":"15%", "date": "17/04/2018"},
  {"name":"rice noodles", "discount":"15%", "date": "28/05/2018"}
   ]
  }
 ]
}
```


Listing 5.1: Snapshot of Clearance Grocery List created on 14 April 2018

5.3 Experiment 1

Persona

Aman is a 27-year-old graduate student at the University of Victoria, Canada. He works part-time at Nibbles and Bytes on campus, and also works as a Teaching Assistant for first year engineering courses. He lives with a roommate in a rented apartment and has limited kitchen space available to store all his groceries. He is fond of eating and misses Indian foods, but does not know their methods of preparation and cannot afford to spend money in an expensive Indian restaurant. Also, after paying all his personal and educational expenses, he is always left with little money available and cannot spend more than $5 a day on food while trying to eat healthy, and as per his taste preferences. Moreover, being a student, he remains quite busy in his studies and does not get time to create a grocery list manually and browse grocery deals before planning his grocery trip. He also rides a bicycle, which limits the number of grocery trips and the amount he can carry.

Based on the information provided by Aman to the SmartGrocer application, following is Aman's context:

Personal Context

From SmartGrocer:
Budget:

- Breakfast: $1
- Lunch: $2
- Dinner: $2

***From* CAPRecipes**:
Diet: Ovo vegetarian
Daily nutrition intake per meal:

- Carbohydrates: 100 grams
- Fat: 20 grams
- Protein: 35 grams
- Calorie: 600 kCal

Allergies: None
Cuisine: Indian
Kitchen Context: Tomatoes, Cilantro, Green pepper, Spinach, Beans, Potatoes, Capsicum, Paneer, Raspberry, Blueberry
Taste Preference access: Yes

Derived Context

The derived context as shown in Listing 5.2 includes personalized coupons in JSON format, which SmartGrocer had recommended to Aman. These coupons were obtained from the Personalized Coupon Engine Component (cf. Chapter 4) by taking context from Aman and the store into account.

```
{ "Data":
 [
  {
  "IngredientsCoupons":
  [
  {"name":"strawberry", "discount":"15%"},
  {"name":"brown rice", "discount":"20%"},
  {"name":"lindt chocolate", "discount":"15%"},
  {"name":"spring onions", "discount":"10%"},
  {"name":"Canned Chickpeas", "discount":"10%"},
  {"name":"Cashews", "discount":"20%"},
  {"name":"Cauliflower", "discount":"15%"},
  {"name":"Mushrooms", "discount":"15%"},
  {"name":"Lemon", "discount":"10%"},
  {"name":"Ginger", "discount":"10%"},
  {"name":"Garlic", "discount":"10%"},
  {"name":"Tofu", "discount":"15%"},
  {"name":"Eggs", "discount":"10%"},
  {"name":"Carrots", "discount":"10%"}
   ]
  }
 ]
}
```


Listing 5.2: Snapshot of the personalized coupons recommended to Aman

5.3.1 Personalized Recipe Recommendations

Figure 5.1 shows personalized lunch recipes recommended by CAPRecipes. These recipes include general information such as name, nutrition details, and used and missing ingredients (i.e., name and quantity) information. However, these personalized recipes do not include the costs of various missing ingredients.

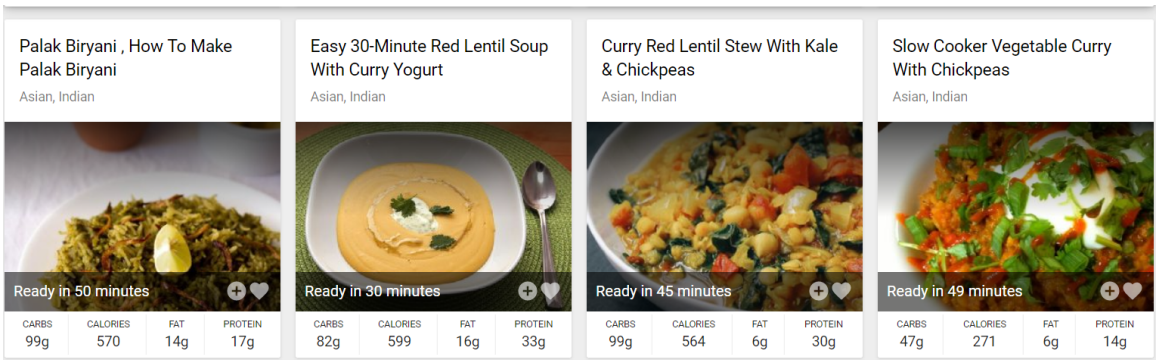


Figure 5.1: Screenshot of recipe recommendations from CAPRecipes

Figure 5.2 depicts how SmartGrocer, in conjunction with the CAPRecipes, recommends personalized lunch recipes when Aman's personalized coupons are not utilized. Here, each recipe cost (i.e., missing ingredients cost) depicted by *MISSING INGREDIENTS* in the snapshot, is calculated using the store context only, and is higher than Aman's budget indicated by the *red* bottom line.

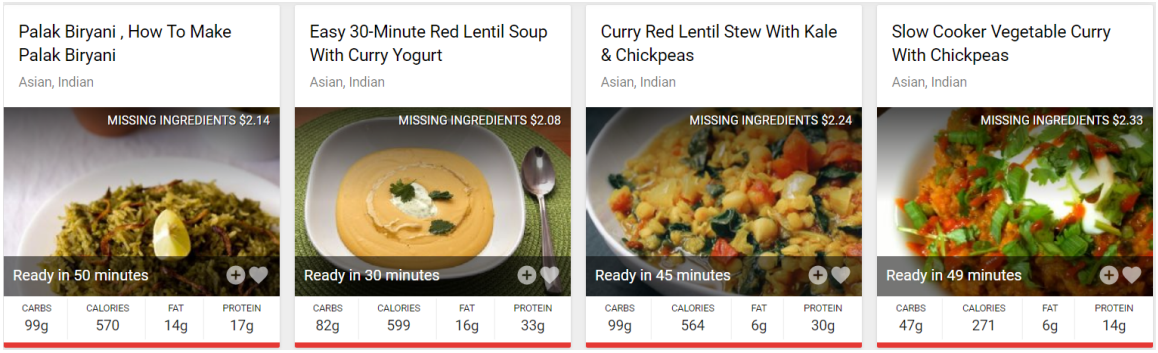


Figure 5.2: Screenshot of recipe recommendations from SmartGrocer without using personalized coupons

Figure 5.3 depicts how SmartGrocer recommends personalized lunch recipes by reducing costs using personalized coupons. Now Aman has recipe choices that fit his budget. The recipes include the cost of missing ingredients before and after applying personalized coupons using our grocery store data, depicted by *MISSING INGREDIENTS* and *MISSING INGREDIENTS AFTER DISCOUNT* respectively. Here, the recipe cost of *Palak Biryani* and *Red Lentil Soup With Curry Yogurt* are reduced, and are now within the Aman's budget, indicated by the *green* bottom line.

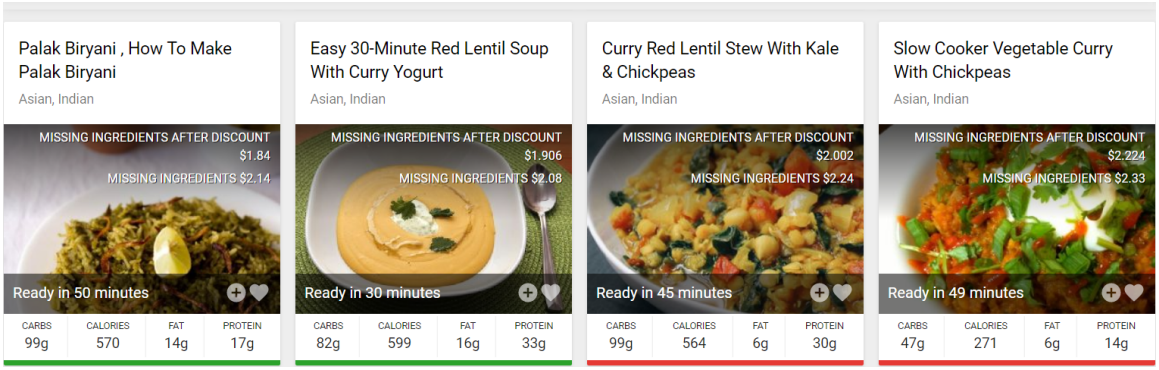


Figure 5.3: Screenshot of recipe recommendations from SmartGrocer using personalized coupons

User Experience

When Aman was asked to provide his opinion on the results of recipe recommendations from both SmartGrocer and CAPRecipes, he found that SmartGrocer results more satisfactory than recommendations from CAPRecipes only. According to him, the cost of missing ingredients shown on each recipe retrieved from SmartGrocer greatly helps him select recipes for preparation. For example, Aman likes *Palak Biryani* which is one of his favorite recipes. But the missing ingredients cost of *Palak Biryani* was not within his budget when recommended by SmartGrocer, without using personalized coupons. With the help of personalized coupons given by SmartGrocer, the cost of the *Palak Biryani* recipe is reduced and now within his budget. So he could now prepare the *Palak Biryani* recipe without any hesitation. On the other hand, he always is in a dilemma while selecting any recipe from CAPRecipes as he had to figure it out by himself whether the recipes are within his budget by calculating recipe costs manually.

Based on his answers, we concluded that he was pleased to get the recipe recommendations from SmartGrocer, which include the recipe cost with and without utilizing personalized coupons because it had solved the issue of his limited budget and the time constraint associated with browsing grocery offers and applying it to recipes manually.

5.3.2 Creation of Grocery List

SmartGrocer had created the grocery list for Aman with the help of the meal plan. The personalized grocery list included ingredient names, coupons and the recipes

which needed the corresponding ingredients. The personalized recipes for breakfast, lunch, and dinner are automatically selected based on the date range of the meal plan. These recipes are according to Aman's taste preferences, health goals, and budget and kitchen context. Moreover, the meal plan also includes Aman's daily savings which is another feature of SmartGrocer. Figure 5.4 depicts the process of creating the grocery list.

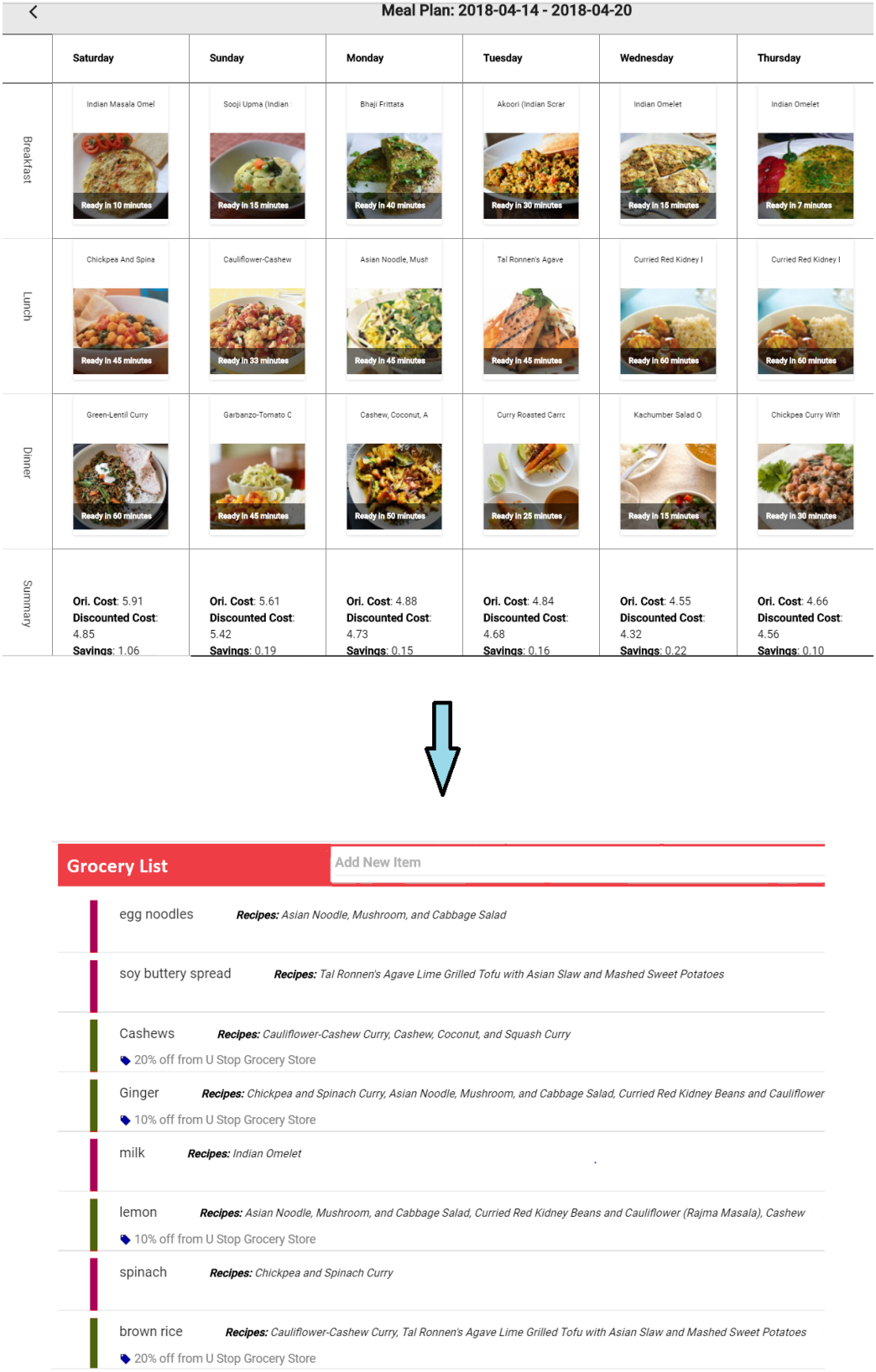


Figure 5.4: Snapshot of creating the Aman's grocery list with the meal plan

Evaluation

Efficiency: 1 step (i.e., creation of grocery list while applying personalized coupons information)

- Usually, Aman selects recipes for each meal time from a third-party meal planning application, looks for grocery offers he could use in his upcoming grocery trip, and then creates a grocery list manually. These steps occupy Aman's time and effort to do it effectively. Here, with the help of SmartGrocer, the whole process of selecting recipes, utilizing grocery coupons and the creation of the grocery list is automated.
- A great deal of time is saved because Aman need not spend time looking out for grocery offers by browsing through different sites or flyers. SmartGrocer generates personalized grocery offers based on Aman's preferences and store context automatically.
- The creation of a grocery list is easy as it involved only one step and therefore is more efficient.

Effectiveness: Highly satisfied with the personalized coupons, which are applied on the grocery list ingredients

- Based on Aman's past purchases, our application recommends personalized coupons on frequently purchased ingredients and the ingredients which could be paired well resulting in new and different recipe recommendations from the system in the future. Aman was quite pleased to see that the coupons were relevant and useful as it was according to his preferences. He did not need to wait for the offers to come up for ingredients which he bought often and to stock unnecessarily in order to fully utilize those offers. Moreover, he could prepare recipes of his choice without thinking about his budget as these personalized coupons were directly applied to the missing ingredients, thereby reducing the overall cost.
- SmartGrocer also recommends additional coupons under the *Recommended Coupons* section as shown in Figure 5.5 for ingredients that are not used in the recipes but their discount coupons are available in the personalized coupons list. Aman was happy to get these coupons because now he can buy other ingredients

which could be used in receiving different personalized recipe recommendations from SmartGrocer in the future, or could be utilized in any regularly prepared recipe.

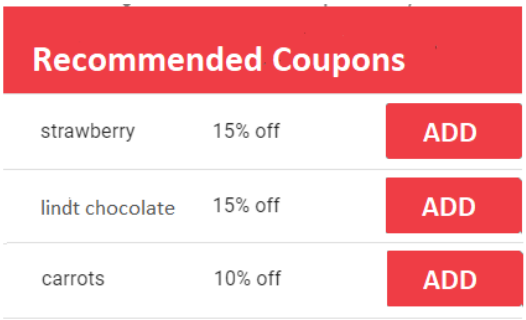


Figure 5.5: Screenshot of recommended coupons from SmartGrocer

User Experience:
After the experiment, Aman was delighted to see a grocery list which not only included the ingredients and coupons associated with it, but also one that provided the recipe information that helped him know which recipe is using which ingredient. He was frustrated because he was not able to search for relevant grocery offers and create the grocery list while maintaining his budget due to his hectic schedule, which had easily been taken into consideration by our system. Aman also added that the recommended personalized coupons resulted in savings as these coupons were for ingredients which he typically buys. On the downside, Aman expressed that SmartGrocer did not update the kitchen context automatically from the grocery purchasing receipts. He had to enter the ingredients information from our application manually.

5.4 Experiment 2

Persona

Lindsay is a forty-year-old woman who lives on her own and loves cooking. She always creates grocery lists manually thinking about the week's meal plan before going grocery shopping as she does not prefer to visit the grocery store more often. However, Lindsay has started developing memory problems as she grows older. Because of this, she sometimes forgets to add ingredients to her grocery list, and as a result, she has to visit the store more than once or change her meal plan for that week, which is quite

frustrating. Also, Lindsay has a limited food budget and therefore wants to utilize grocery offers to minimize her expenditures. But it is infeasible for her to search for relevant grocery offers every time as it takes too much of her time. Based on the information provided by her, the following context is gathered from our system.

Personal Context

From SmartGrocer:
Budget:

- Breakfast: $1.5
- Lunch: $3
- Dinner: $3

***From* CAPRecipes**:
Diet: Non vegetarian
Daily nutrition intake per meal:

- Carbohydrates: 150 grams
- Fat: 30 grams
- Protein: 50 grams
- Calorie: 850 kCal

Allergies: Nuts
Cuisine: Korean, Chinese, Thai
Kitchen Context: Beef, Brown rice, Chicken, Black Beans, Kale, Quinoa, Avocado, Lemon, Ginger, Cucumber, Garlic, Mushrooms, Broccoli, Lettuce
Social Media permission access: Yes

Derived Context

The derived context is the personalized coupons recommended to Lindsay, which is arranged in the JSON format depicted in Listing 5.3.

```
{
 "Data":
 [
  {
  "IngredientsCoupons": [
  {"name":"carrots", "discount":"10%"},
  {"name":"spring onions", "discount":"10%"},
  {"name":"Cauliflower", "discount":"15%"},
  {"name":"Salmon", "discount":"15%"},
  {"name":"Eggs", "discount":"10%"},
  {"name":"Peas", "discount":"10%"},
  {"name":"Shrimp", "discount":"15%"},
  {"name":"Coconut Milk", "discount":"15%"},
  {"name":"Pork Loin", "discount":"15%"},
  {"name":"Cabbage", "discount":"15%"},
  {"name":"Gruyere", "discount":"15%"},
  {"name":"Cardamom", "discount":"20%"},
  {"name":"Bacon", "discount":"15%"},
  {"name":"rice noodles", "discount":"15%"},
  {"name":"Tofu", "discount":"15%"}
  ]
   }
  ]
}
```


Listing 5.3: Snapshot of the personalized coupons recommended to Lindsay

5.4.1 Personalized Recipe Recommendations

When Lindsay requested dinner recipes from our application and CAPRecipes, the systems recommended recipes with and without including recipe costs respectively.

Figure 5.6 displays the personalized dinner recipes from CAPRecipes. These recipes were recommended based on Lindsay's taste preferences, health goals, and her kitchen context.

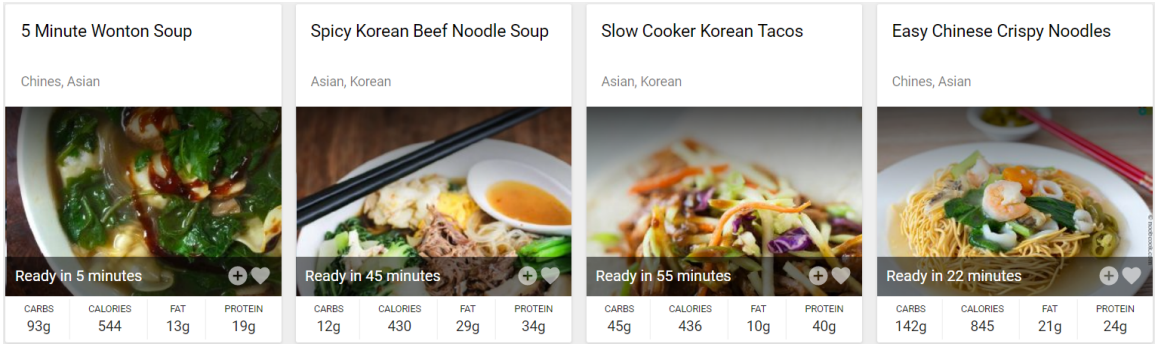


Figure 5.6: Screenshot of recipe recommendations from CAPRecipes

Figure 5.7 shows personalized dinner recipes from SmartGrocer where each recipe includes the cost in terms of the original missing ingredient cost without using personalized coupons. Here, only one recipe is within Lindsay's dinner budget (i.e., $3).

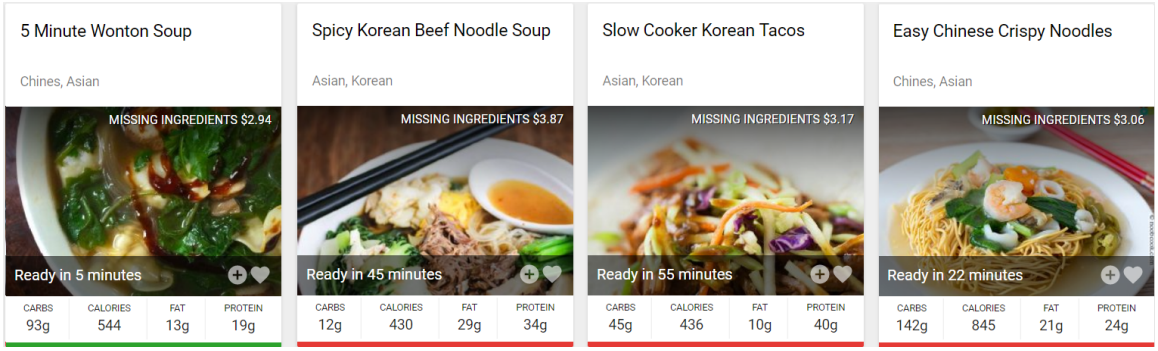


Figure 5.7: Screenshot of recipe recommendations from SmartGrocer without using personalized coupons

Figure 5.8 shows personalized dinner recipes from SmartGrocer with personalized coupons applied to each recipe. The two cost values are displayed on each recipe: one is the original missing ingredients cost (i.e.,*MISSING INGREDIENTS*) and another is the reduced missing ingredients cost (i.e., *MISSING INGREDIENTS AFTER DISCOUNT*), which is calculated by utilizing personalized coupons. Now, two more recipes from the recommended recipe list are within her food budget.

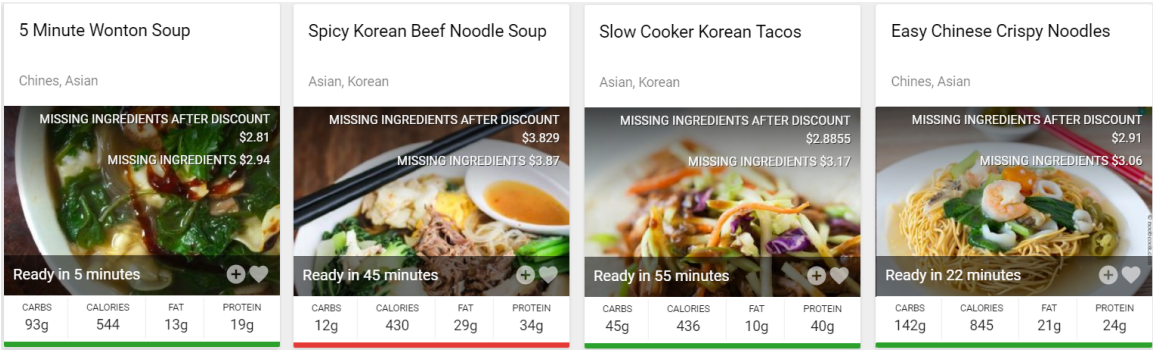


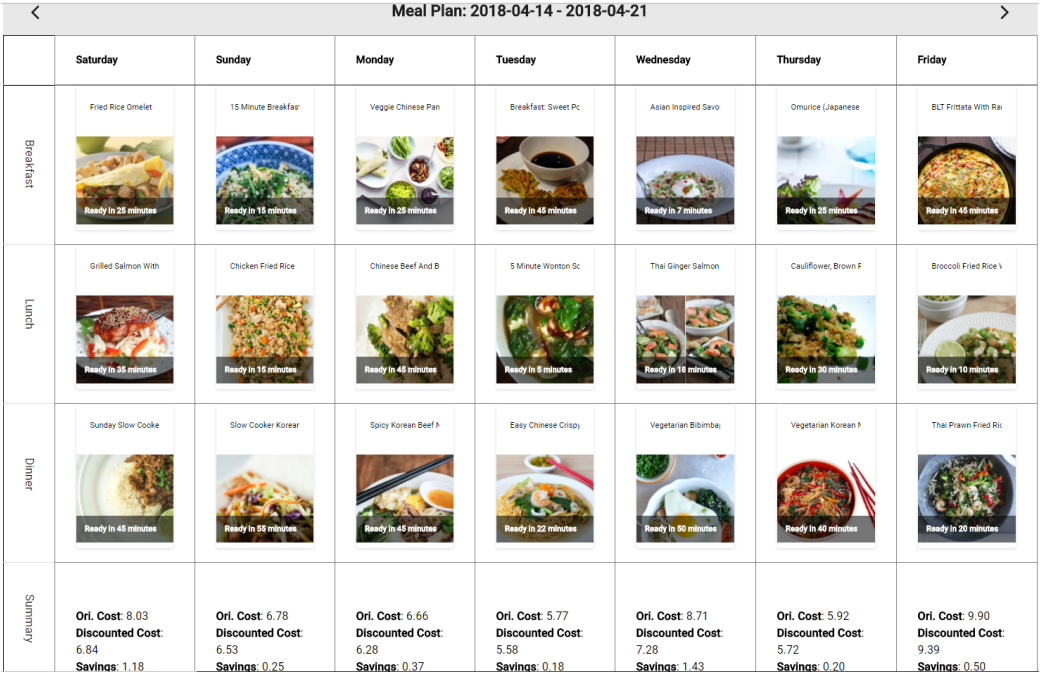
Figure 5.8: Screenshot of recipe recommendations from SmartGrocer using personalized coupons

User Experience

When we asked Lindsay to provide feedback on her recipe recommendations from both the SmartGrocer and CAPRecipes, she said that showing the cost of each recipe is a nice feature of SmartGrocer as it assisted her in choosing recipes effectively. Moreover, with SmartGrocer she has now more recipes that are within her budget thanks to personalized coupons. Based on her answers, we found that she was pleased to use our system as it encourages her to prepare meals within her budget.

5.4.2 Creation of Grocery List

Lindsay's grocery list was created from our application which includes personalized coupons and recipe information for each ingredient as depicted in Figure 5.9. This list was generated through the week-long meal plan she created.



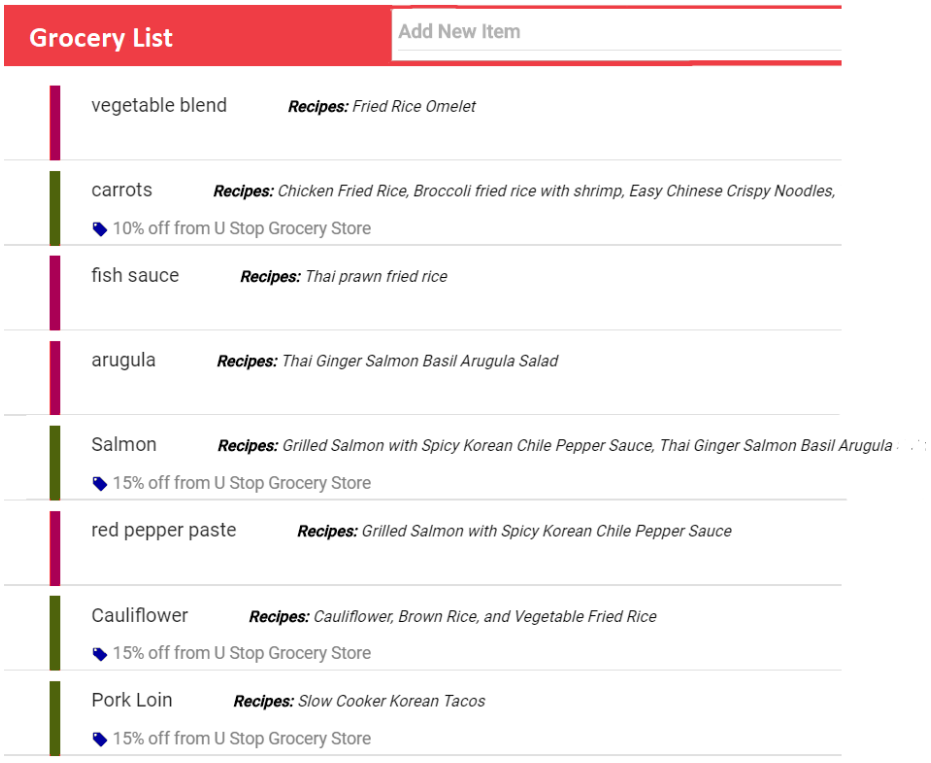


Figure 5.9: Snapshot of creating the Lindsay's grocery list with the meal plan

Evaluation

Efficiency: 1 step (i.e., the creation of grocery list while applying personalized coupons information)

- Ordinarily, Lindsay would go to third-party online grocery store sites, such as Thrifty Foods or Whole Foods to select recipes and create a grocery list by adding all the missing ingredients into her shopping list directly. But, she would have to look for relevant grocery offers separately to minimize her expenditures on groceries, which is a time-consuming process. With the help of SmartGrocer, the process of selecting the recipes, applying the grocery coupons and the creation of a grocery list is automated and accomplished in one step.

- A great deal of time is saved as Lindsay need not spend time browsing grocery offers manually from separate applications because our system recommends personalized coupons, which are dynamically generated and automatically applied to the recipes recommended from our system.

- The creation of grocery list is easy and quick as it takes only one step to get the list.

Effectiveness: Highly satisfied with the personalized coupons, which are applied to the grocery list ingredients

- When using SmartGrocer, we noted that Lindsay's preferences, in terms of her past purchasing histories, are taken into consideration while recommending personalized coupons to her. She is happy with the personalized coupons presented and utilized automatically by our system when recommending recipes within her budget. She did not need to browse or wait for the relevant grocery offers during the selection of recipes.

- SmartGrocer also recommended additional coupons under the *Recommended Coupons* section as depicted in Figure 5.10 for the ingredients, which are available on the personalized coupons list but not used in the recipes. Lindsay is happy to get coupons for other ingredients and will use these coupons to get different recipe recommendations from SmartGrocer in the future.

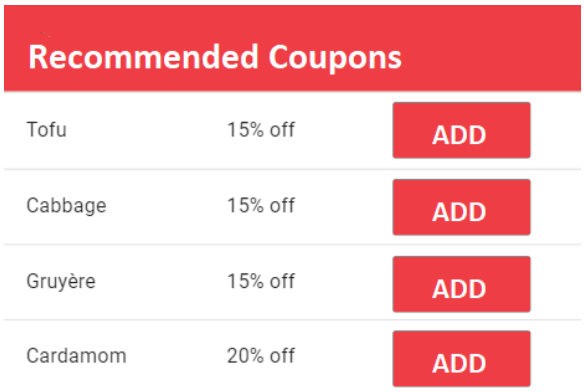


Figure 5.10: Screenshot of recommended coupons from SmartGrocer

User Experience:
After the experiment, we asked her opinion of our application. Based on her answers, we concluded that she was pleased with the experience, and would prefer to use SmartGrocer over other applications. She determined that SmartGrocer was not only helping her in creating the grocery list with ease but also improved her savings with the help of personalized coupons retrieved based on her dynamic context (i.e., past purchasing histories and budget) and store context (i.e., clearance grocery list). However, Lindsay experienced that our application does not consider other names of the same ingredient when then do not get added to her grocery list. For example, a recipe has *scallions* as a missing ingredient, but the store has *scallions* under different names, such as *spring onions* or *green onions* — our application does not handle such naming variances.

5.5 Grocery Store Analysis

The store's *Clearance Grocery List* includes ingredients that are soon-to-expire or on sale. This list includes both perishable ingredients that expire in four days and non-perishable ingredients. SmartGrocer determined Aman's and Lindsay's frequently purchased items and the paired ingredients list from their past purchase histories. Then our application recommends coupons to them after comparing their frequent and paired ingredients list with the store's *Clearance Grocery List* as depicted in the Figure 5.11. Our application considers the expiry date of ingredients while recommending offers to our volunteers, which encourages the store to sell ingredients before

they expire. This way of delivering offers is not only saving volunteers' money, but also promoting the reduction of food waste by the store.

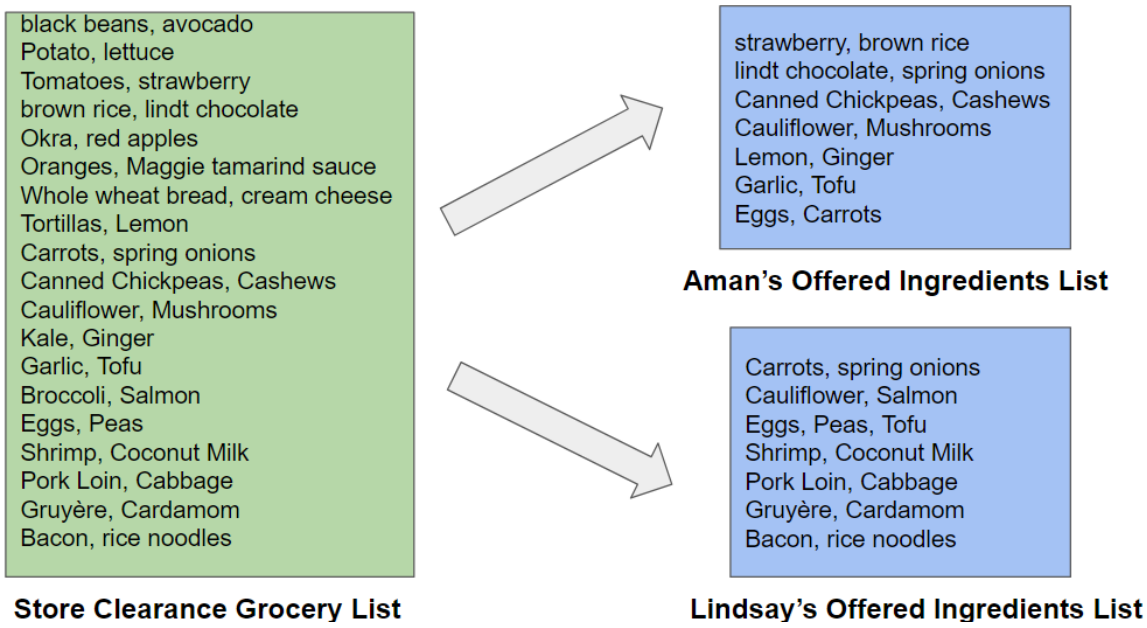


Figure 5.11: Aman and Lindsay's offered ingredient list from store clearance grocery list

5.6 Summary

This chapter evaluated the SmartGrocer performance on the basis of efficiency, effectiveness, and user experience with the help of experiments conducted with two volunteers. The results and volunteer feedback helped us identify both positive and negative aspects of SmartGrocer. Our preliminary experiments indicate that SmartGrocer eliminates repetitive steps while creating a shopping list, thereby augmenting the user experience.

Chapter 6

Conclusions

This chapter summarizes the work performed to answer the research questions formulated in the introduction. We also illustrate the achievements and identify potential areas for future work.

6.1 Summary

This thesis presented the prototype application SmartGrocer, a Context-Aware Personalized Grocery System. The motivation for this research came from the identification of several challenges faced by users during grocery shopping, and food waste, a primary concern of the world. Food waste occurs at both personal (i.e., in homes) and business (i.e., in stores) levels and our system focused on providing solutions for the latter. Usually, users face various challenges before they plan their grocery trip, as follows:

- Selection of recipes according to budget, along with taste preferences and health restrictions as they prefer recipes within their food budget in their everyday life.
- Manual creation of grocery lists after users select recipes for their meal plan, considering their food budget and grocery offers if applicable.
- General grocery offers provided by stores which might not be useful for their upcoming grocery shopping trips.

The proposed application overcomes these challenges by leveraging user and store contexts. It utilizes user buying behavior through their purchase histories and food

budget, as well as the stock of ingredients that are soon to expire or on sale that a store wants to move, by recommending personalized coupons. This new way of providing personalized coupons to users promotes proper handling of excess inventory by the store and reducing the amount of perishables that ultimately go to waste. Personalized coupons are applied directly to missing ingredients of recipes, thereby reducing the overall cost of recipes resulting in recipes that are not within the user's budget. The application has also automated the process of creating the grocery list that starts with the selection of recipes, includes utilization of grocery coupons, and ends with the generation of a grocery list that includes missing ingredient names, recipes which use these ingredients and coupon information.

Chapter 2 presented the literature review on consumer buying behavior to help understand the significance of the 4 Ps (i.e., Product, Price, Place, and Promotion) of the marketing mix that influence user choice of grocery store. We also explored the major concepts of context-aware computing and the importance of personalizing the context and algorithms used to achieve personalization as related to our research. We illustrated various examples of applications related to grocery shopping and their limitations in providing better personalized experiences to users.

Chapter 3 demonstrated the use of RESTful APIs, such as Spoonacular and Foodpairing, and algorithms for implementing different functionalities of SmartGrocer. We also explained a natural language processing task that is used to match ingredient names gathered from various sources to the most relevant match available in the grocery database by using the Part-Of-Speech (POS) tagging technique.

Chapter 4 presented the architectural design of SmartGrocer including components of the system and their functionalities. We also explained the interface design of our system to illustrate the key features and functionalities of our proposed application.

Chapter 5 presented the two case studies conducted with the help of two volunteers. The volunteers assessed our application in terms of efficiency, effectiveness and user experience. They shared their experiences using SmartGrocer and also suggested a few areas of improvement, discussed in future work.

6.2 Contributions

This section summarizes the main contributions of the thesis:

- Concept and implementation of automating the process of creating the grocery list by exploiting context.
  SmartGrocer leverages user contexts (i.e., past purchasing details and budget), store context (i.e., grocery stock information) and personalized recipe recommendations from CAPRecipes to create grocery lists automatically.

- We developed the Personalized Coupon Engine (PCE) algorithm that exploits user purchase histories, the Foodpairing API, and soon-to-expire or new promotional grocery ingredient lists retrieved from the store to recommend personalized coupons. The coupons are directly applied to recipes, thereby reducing their cost. Coupon lists are dynamically updated based on changes occurring in user or store contexts.

- We explored the Foodpairing API to recommend coupons for ingredients which users do not buy often, but these ingredients can be paired well with their frequently purchased ingredients. This way of recommendation encourages users to buy ingredients other than frequently purchased ingredients resulting in different recipe recommendations.

- We promoted the saving of user money and the reduction of food waste in stores, as personalized coupons are generated based on purchase histories and the list of soon-to-expire and new promotional ingredients available from the grocery store.

- We explored the Natural Language Toolkit (NLTK) package and developed an algorithm called the POS Tagging Mean Probability algorithm, to replace different naming structures of an ingredient retrieved from multiple web sources with the most relevant match obtained from the grocery database.

- We developed SmartGrocer in conjunction with CAPRecipes as a recipe recommendation system that recommends recipes not only based on user taste and health but also on budget by displaying the cost of each recipe.

6.3 Future Work

SmartGrocer is designed to demonstrate the use of context in the field of grocery shopping which benefited both users and store businesses. This section summarizes

ideas that can further improve the efficiency and effectiveness of the application, and augment the user experience as outlined below:

- SmartGrocer, currently uses user preferences derived from their purchase history, and the synergy of frequently purchased ingredients with other ingredients, to recommend personalized coupons. Our application could implement some collaborative filtering techniques to recommend personalized coupons based on other users with similar buying behavior.

- FridgeCam is the world's first wireless fridge camera that allows users to see the contents of their refrigerator from anywhere. We could use this device with our application to track the ingredients that are being used in some other preparation which are not recommended by our application as FridgeCam takes a photo every time users close their fridge door [Hom18].

- One problem that users might face is locating an item within the store. SmartGrocer could arrange a grocery list according to aisle number, food category, and by using path optimization algorithms so that users can travel the shortest path through the store to pick up all the items on their list.

- Currently, SmartGrocer is a desktop application, but it could be implemented as a mobile application.

- SmartGrocer does not consider other names of ingredients while creating the grocery list or providing personalized coupons to users. For example, a recipe has *scallions* as a missing ingredient but the store has *scallions* under different names (i.e., *spring onions*, *green onions*) — our application does not handle such naming variances. So, to resolve the naming variances of ingredients, we could create a separate database that has other names of most of the ingredients available in stores, so that our application would be able to create a grocery list or provide personalized coupons to users more effectively.

- SmartGrocer uses user's past purchase histories given by the store to recommend personalized coupons. We could use machine learning techniques to analyze user's purchases and usage of ingredients in the preparations that are recommended by the SmartGrocer for recommending personalized coupons.

- SmartGrocer could be extended to track the user's purchases for non-food items such as toiletry, for recommending personalized coupons for these items.

Glossary

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

GC-MS Gas Chromatography–Mass Spectrometry (GC-MS) is an analytic method that helps in identifyting different substances within a test sample.

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

HCI Human Computer Interaction (HCI) is an area of research that focuses on the interfaces between people (users) and computers.

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

JSON JavaScript Object Notation (JSON) for storing and exchanging data.

Marketing mix Marketing mix is the set of marketing tools that the firm uses to influence consumers to purchase their products.

NLTK The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming 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.

Smart home Smart home is building automation for homes.

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

UML Unified Modeling Language (UML) is a modeling language in the field of software engineering.

URI Uniform Resource Identifier (URI) is string of characters that identifies a logical or physical resource.

XML eXtensible Markup Language for storing and exchanging data.

References[1]

[Abr18] M. Abraham. *Profiting from personalization.* May 2017 [Accessed on 06-January-2018]. URL. 3

[Ali18] E. Alini. *How much of your budget should you spend on groceries? - National — Globalnews.ca.* May 2017 [Accessed on 05-January-2018]. URL. 2

[App51] W. Applebaum. Studying customer behavior in retail stores. *Journal of Marketing*, 16(2):172–178, 1951. 7, 10, 11

[BA18] S. Ben-Achour. *What do stores do with unsold merchandise?* March 2014 [Accessed on 06-January-2018]. URL. 3

[BBC97] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, 1997. 16

[BF15] J. Burrow and A. Fowler. *Marketing.* Cengage Learning, 2015. URL. 7

[BHBR17] M. Buisman, R. Haijema, and J. Bloemhof-Ruwaard. Discounting and dynamic shelf life to reduce fresh food waste at retailers. *International Journal of Production Economics*, 2017. 3

[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. 16

[Blu18] Bluespeed. *7 Greatest advantages of smart-home automation.* June 2016 [Accessed on 04-January-2018]. URL. 1

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

[Car18] B. Carter. *Coupon statistics: The ultimate collection.* November 2017 [Accessed on 05-January-2018]. URL. 3

[CBSD01] M. Connors, C. A. Bisogni, J. Sobal, and C. M. Devine. Managing values in personal food systems. *Appetite*, 36(3):189–200, 2001. 2

[Cle18] Cleverism. *Understanding the marketing mix concept — 4Ps.* January 2017 [Accessed on 20-January-2018]. URL. 9

[DAS01] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, 2001. 16, 17

[Das18] S. Das. *Beginners guide to learn about content based recommender engine.* August 2015 [Accessed on 21-January-2018]. URL. 21

[DAW98] A. K. Dey, G. D. Abowd, and A. Wood. Cyberdesk: A framework for providing self-integrating context-aware services. *Knowledge-Based Systems*, 11(1):3–13, 1998. 16

[DC05] P. R. Darke and C. M. Chung. Effects of pricing and promotion on consumer perceptions: it depends on how you frame it. *Journal of Retailing*, 81(1):35–47, 2005. 3

[Dey01] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001. 16

[Eat18] EatLove. *EatLove Announces Collaboration with AmazonFresh.* November 2017 [Accessed on 19-April-2018]. URL. 3

[Eng97] J. Engel. Blackwell, rd & miniard, pw (1995), consumer behavior. *New York: Dryden Press. Farr, A., & Hollis*, (1997):23–36, 1997. 9

[Epi18] Epicurious. *The best grocery apps — Epicurious.com.* January 2017 [Accessed on 22-January-2018]. URL. ix, 22, 23

[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. 38, 39

[FAH15] M. Familmaleki, A. Aghighi, and K. Hamidi. Analyzing the influence of sales promotion on customer purchasing behavior. *International Journal of Economics & Management Sciences*, 4(4):1–6, 2015. 11

[fB18] R. for Business. *Public relations - Benefits, goals of public relations*. January 2017 [Accessed on 20-January-2018]. URL. 12

[Fis12] G. Fischer. Context-aware systems: the 'right' information, at the 'right' time, in the 'right' place, in the 'right' way, to the 'right' person. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI 2012)*, pages 287–294. ACM, 2012. 22

[Fis18] I. V. Fishchuk. *How to Develop a Grocery Shopping List App? — MLS-Dev*. August 2016 [Accessed on 21-April-2018]. URL. 22

[Foo18a] Foodpairing. *Foodpairing API — Apiary*. January 2017 [Accessed on 01-February-2018]. URL. 32, 33

[Foo18b] FoodPairing. *Science behind — Foodpairing*. December 2017 [Accessed on 01-February-2018]. URL. ix, 30, 31, 32

[Fre03] S. A. French. Pricing effects on food choices. *The Journal of Nutrition*, 133(3):841S–843S, 2003. 2

[Fre18] Freshome. *Top 10 benefits of automating your home*. January 2017 [Accessed on 04-January-2018]. URL. 1

[FT00] R. T. Fielding and R. N. Taylor. *Architectural Styles and the Design of Network-Based Software Architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000. 13

[Gar18] L. Garibian. *Customer behavior — Personalized marketing drives buyer readiness and sales : MarketingProfs Article*. March 2013 [Accessed on 06-January-2018]. URL. 3

[GS04] J. Gebauer and M. J. Shaw. Success factors and impacts of mobile business applications: results from a mobile e-procurement study. *International Journal of Electronic Commerce*, 8(3):19–41, 2004. 1

[GW16] R. Gasior and K. Wojtycza. Sense of smell and volatile aroma compounds and their role in the evaluation of the quality of products of animal origin– a review. *Annals of Animal Science*, 16(1):3–31, 2016. 31

[Haa18] S. Haan. *Grocery shopping list — Healthy living at WomansDay.com*. February 2011 [Accessed on 05-January-2018]. URL. 2

[HC03] C. K. Hess and R. H. Campbell. An application of a context-aware file system. *Personal and Ubiquitous Computing*, 7(6):339–352, 2003. 16

[Hom18] S. Home. *Smarter FridgeCam (ORDER NOW)*. January 2017 [Accessed on 23-March-2018]. URL. 85

[Inv18] Investopedia. *Four Ps*. January 2018 [Accessed on 20-January-2018]. URL. 11

[Ivy08] J. Ivy. A new higher education marketing mix: the 7ps for mba marketing. *International Journal of Educational Management*, 22(4):288–299, 2008. 10

[Jai18] H. Jain. CAPRecipes: A context-aware personalized recipes recommender for healthy and smart living. Master's Thesis, Department of Computer Science, University of Victoria, 2018. ix, 5, 48, 58

[JKS00] K. H. Joo, T. Kinoshita, and N. Shiratori. Agent-based grocery shopping system based on user's preference. In *Seventh International Conference on Parallel and Distributed Systems ,2000*, pages 499–505. IEEE, 2000. ix, 28

[JSK10] G. Jawaheer, M. Szomszor, and P. Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems*, pages 47–51. ACM, 2010. 20

[Kan18] H. Kanapi. *15 Online Shopping Statistics That You Should Know*. July 2017 [Accessed on 19-April-2018]. URL. 1

[Kaz18] D. Kazovskaya. *Digital transformation: 5 Ways for grocery stores*. February 2016 [Accessed on 28-February-2018]. URL. 1

[Ken18] J. Kennedy. *How mobile apps are changing the retail Industry - Carson Shopify*. July 2016 [Accessed on 04-January-2018]. URL. 1

[KT03] D. Kelly and J. Teevan. Implicit feedback for inferring user preference: a bibliography. In *Acm Sigir Forum*, volume 37, pages 18–28. ACM, 2003. 20

[Lai14] K. Laine. The factors influencing the choice of grocery store among finnish consumers. 2014. ix, 9, 10, 12

[LCL12] H. Li, F. Cai, and Z. Liao. Content-based filtering recommendation algorithm using hmm. In *Fourth International Conference on Computational and Information Sciences (ICCIS 2012)*, pages 275–277. IEEE, 2012. 21

[Lie18] E. Liem. *What will grocery shopping look like in the future? — Food Dive*. July 2017 [Accessed on 05-January-2018]. URL. 2

[LMSR09] R. Lucena Matamalas and M. Santandreu Ramos. Marketing strategic of supermarkets, 2009. 8, 10

[LS00] H. Lieberman and T. Selker. Out of context: Computer systems that adapt to, and learn from, context. *IBM Systems Journal*, 39(3.4):617–632, 2000. 15

[LWM+15] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang. Recommender system application developments: A survey. *Decision Support Systems*, 74:12–32, 2015. 1

[Mar18] Marketplace. *Recipe — Food — Nutrition API documentation*. January 2017 [Accessed on 01-February-2018]. URL. ix, 36

[Met18] Metro. *My tastes — Metro*. January 2017 [Accessed on 22-January-2018]. URL. ix, 3, 23, 24, 25

[MM11] N. Muzondo and E. Mutandwa. The seven ps of marketing and choice of main grocery store in a hyperinflationary economy. *Contemporary Marketing Review*, 1(9):01–18, 2011. 8, 9, 10, 11, 12

[Mob07] B. Mobasher. Data mining for web personalization. In *The adaptive web*, pages 90–135. Springer, 2007. 18

[NDA12] F. Nayebi, J.-M. Desharnais, and A. Abran. The state of the art of mobile application usability evaluation. In *25th IEEE Canadian Conference on Electrical & Computer Engineering (CCECE 2012)*, pages 1–4. IEEE, 2012. 1

[Nik18] Nikke. *Marketing Mix Evolution: 4P's to SAVE - Brand2Love Marketing*. July 2015 [Accessed on 15-June-2018]. URL. 9

[Niu09] T. Niu. *Strategies for success in the e-Grocery Industry*. Rochester Institute of Technology, 2009. Accessed from URL. 1

[Nlt18] Nltk. *Categorizing and tagging words*. January 2017 [Accessed on 02-February-2018]. URL. 38

[Ora18] Oracle. *What are RESTful web services? — The Java EE 6 tutorial*. January 2013 [Accessed on 02-February-2018]. URL. 13

[Pla18] F. M. Plan. *4 Things you must do before you go to the grocery store*. January 2017 [Accessed on 05-January-2018]. URL. 2

[Res18] A. Resources. *Output formats*. February 2012 [Accessed on 02-February-2018]. URL. 14

[RG08] A. Rajak and M. K. Gupta. Association rule mining: Applications in various areas. In *Proceedings of International Conference on Data Management, Ghaziabad, India*, pages 3–7, 2008. 19

[Rod18] A. Rodriguez. *RESTful web services: The basics*. February 2015 [Accessed on 02-February-2018]. URL. 13

[RRH01] O. W. Rahlff, R. K. Rolfsen, and J. Herstad. Using personal traces in context space: Towards context trace technology. *Personal and Ubiquitous Computing*, 5(1):50–53, 2001. 16

[San18] K. Sandoval. *Defining Stateful Vs Stateless Web Services — Nordic APIs*. May 2017 [Accessed on 20-April-2018]. URL. 14

[Sat01] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001. 15

[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. 2, 15, 16

[SBG99] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, 1999. 16

[SKKR01] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web (WWW 2001)*, pages 285–295. ACM, 2001. 21

[SLND12] S. Sehic, F. Li, S. Nastic, and S. Dustdar. A programming model for context-aware applications in large-scale pervasive systems. In *IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2012)*, pages 142–149. IEEE, 2012. 15

[SLP04] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop Proceedings*, 2004. 17, 18

[SMS08] P. Shoval, V. Maidel, and B. Shapira. An ontology-content-based filtering method. 2008. 20

[SNH03] S. Shekar, P. Nair, and A. S. Helal. igrocer: a ubiquitous and pervasive smart grocery shopping system. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC 2003)*, pages 645–652. ACM, 2003. ix, 26, 27

[Spo18] Spoonacular. *Food and Recipe API*. January 2017 [Accessed on 01-February-2018]. URL. 36

[ST94] B. N. Schilit and M. M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994. 16

[Sta18] Statista. *U.S. consumers: Online grocery shopping statistics & facts — Statista*. January 2017 [Accessed on 21-January-2018]. URL. 22

[Ste18] M. Sternberg. *How content personalization is changing grocery marketing*. August 2017 [Accessed on 04-January-2018]. URL. 2

[TACH13] R. Teymourzadeh, S. A. Ahmed, K. W. Chan, and M. V. Hoong. Smart gsm based home automation system. In *IEEE Conference on Systems, Process & Control (ICSPC 2013)*, pages 306–309. IEEE, 2013. 1

[Tar18] M. Tartalio. *Consumer food trends: Tapping into personalized foods*. January 2018 [Accessed on 06-January-2018]. URL. 3

[Tea18] F. H. Team. *10 Expert tips to grocery shopping on a budget – Health essentials from Cleveland clinic*. March 2016 [Accessed on 05-January-2018]. URL. 2

[Tip18] S. S. Tips. *Concentrate on the big three expenses of your budget - Super saving tips*. March 2018 [Accessed on 05-January-2018]. URL. 2

[TSK05] P.-N. Tan, M. Steinbach, and V. Kumar. Association analysis: basic concepts and algorithms. *Introduction to Data Mining*, pages 327–414, 2005. 20

[UGS14] N. Uzea, M. Gooch, and D. Sparling. *Developing an industry led approach to addressing food waste in Canada*. Provision Coalition, 2014. 3, 4

[UKE18] UKEssays. *Importance of consumer behaviour*. May 2017 [Accessed on 06-January-2018]. URL. 4, 7

[Vaq18] M. Vaqqas. *RESTful web services: A tutorial — Dr Dobb's*. September 2014 [Accessed on 02-February-2018]. URL. 13

[WHH11] B. Weber, S. Herrlein, and G. Hodge. The challenge of food waste. *Planet Retail*, 2011. 3

[WL04] S. S. Weng and M. J. Liu. Personalized product recommendation in e-commerce. In *IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE 2004)*, pages 413–420. IEEE, 2004. 2

Appendix A

Source Code

Listing A.1: Python code to get the personalized recipe list with the costs information. The explanation of this code is in the Section 3.2.

```
# Implemenation to get the "price" of each missing ingredient of recipes
# Implementation to get the "missedIngredientsCost" and "
    missedIngredientsCostAfterDiscount" of each recipe
#!/usr/bin/python27

# Usage for getting recipelist: http://localhost:8020/SmartGrocer/recipelist?type=
    main%20course&number=20&cuisine=indian,mexican&couponFile=<filename>
# Usage for getting cookingstep: http://localhost:8020/SmartGrocer/cookingsteps?id
    =584043


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
import urllib

class RecipeRunner(Thread):
  def __init__(self, queue):
    Thread.__init__(self)
    self.queue = queue

  def run(self):
    while True:
      # get the work from the queue and expand the tuple
      recommendedRecipeList = self.queue.get()
```

```
31        reclistGenWithCost(recommendedRecipeList)
32        self.queue.task_done()
33
34  app = Bottle()
35
36  @app.hook('after_request')
37  def enable_cors():
38    """
39    You need to add some headers to each request.
40    Don't use the wildcard '*' for Access-Control-Allow-Origin in production.
41    """
42    response.headers['Access-Control-Allow-Origin'] = '*'
43    response.headers['Access-Control-Allow-Methods'] = 'PUT, GET, POST, DELETE, OPTIONS
        '
44    response.headers['Access-Control-Allow-Headers'] = 'Origin, Accept, Content-Type, X
      -Requested-With, X-CSRF-Token'
45
46  @app.route('/SmartGrocer/<name>', method=['GET'])
47  def recdata(name):
48    # To generate recipe list with "price" attribute for each missing ingredient and "
        missedIngredientsCost" of a recipe
49    if name == "recipelist":
50      # reading query parameters from GET request
51      foodtype = request.query.type
52      recipeCount = request.query.number
53      cuisineinfo = request.query.cuisine
54      coupFile = request.query.couponFile
55      userHealthAcess = True
56      userKitchenContextAcess = True
57      userSocialMediaAcess = True
58      userAvailableIngredientExpiryAcess = True
59      listrecm = {}
60      # calling CAPRecipes to get the personalized recipe recommendations
61      listrecm  = urllib.urlopen("http://localhost:3020/CAPRecipes/recipelist?type="+
        foodtype+"&number="+recipeCount+"&cuisine="+cuisineinfo+"&userHealthContext="+str
        (userHealthAcess)+"&userKitchenContext="+str(userKitchenContextAcess)+"&
        userTasteContext="+str(userSocialMediaAcess)+"&userAvailableIngredientExpiry="+
        str(userAvailableIngredientExpiryAcess))
62
63      with io.open('SG_recipes.json', 'w', encoding='utf-8') as f:
64        f.write(unicode(listrecm.read()))
65
66      with open('SG_recipes.json') as data_file:
67        rdata = json.load(data_file)
68
69      # create a queue to communicate with the worker threads
70      queue = Queue()
71      # create 10 worker threads for determining the price of missing ingredients
72      for x in range(10):
73        worker = RecipeRunner(queue)
74        # setting daemon to True will let the main thread exit even though the workers
        are blocking
75        worker.daemon = True
```

```
76        worker.start()
77      for recipes in rdata["data"]:
78        queue.put(recipes)
79
80      queue.join()
81      with io.open('SG_recresponse.json', 'w', encoding='utf-8') as f:
82        f.write(unicode(json.dumps(rdata,ensure_ascii=False)))
83
84      # calling function to apply persoonalized coupons on recipe missing ingredients
85      listrecmnew = reclistGenWithCoupons(rdata, coupFile)
86      return listrecmnew
87    # To generate recipe instruction page
88    elif name == "cookingsteps":
89      recipeid = request.query.id
90      instList = {}
91      # calling CAPRecipes to get the recipe instruction page
92      instList  = urllib.urlopen("http://localhost:3020/CAPRecipes/cookingsteps?id="+
      recipeid)
93
94      return instList.read()
95
96  # To calculate similarity between two words
97  def similar(a, b):
98      return SequenceMatcher(None, a, b).ratio()
99
100 # To apply personalized coupons to recipes
101 def reclistGenWithCoupons(recipesData, cfile):
102   with open(cfile) as data_file:
103     coupons = json.load(data_file)
104   for eachRecipe in recipesData["data"]:
105     missedIngredientsCost = eachRecipe["missedIngredientsCost"];
106     for eachMissedIngredient in eachRecipe["missedIngredients"]:
107       for IngrCoupons in coupons["Data"]:
108         for eachCoupon in IngrCoupons["IngredientsCoupons"]:
109           # checking if an ingredient name is similar to ingredient name present in "
      Clearance Grocery List"
110           if similar(eachCoupon["name"],eachMissedIngredient["name"]) >0.7:
111             eachIngrPrice = float(eachMissedIngredient["price"][1:])
112             eachIngrReducedPrice = (float(eachCoupon["discount"])/100) *
      eachIngrPrice
113             missedIngredientsCost = missedIngredientsCost - eachIngrReducedPrice
114     eachRecipe["missedIngredientsCostAfterDiscount"] = round(missedIngredientsCost,2)
115   return recipesData
116
117 # To calculate the cost of missing ingredients of a recipe
118 def reclistGenWithCost(recipeData):
119   price = 0.0
120   for missedattr in recipeData["missedIngredients"]:
121     length = len(missedattr)
122     missedattr['amount'] = round(missedattr['amount']/recipeData['servings'],2)
123     ingredientinfo = missedattr['name'] + " " + str(round(missedattr['amount'],2)) +
      " " + missedattr['unitLong']
124     # calling Spoonacular Grocery API
```

```
    mresponse = unirest.post("https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/visualizePriceEstimator",
    headers={
    "X-Mashape-Key": "API_KEY",
    "Accept": "text/html",
    "Content-Type": "application/x-www-form-urlencoded"
    },
    params={
    "defaultCss": True,
    "ingredientList": ingredientinfo,
    "mode": 1,
    "servings": 1,
    "showBacklink": True
    }
    )
    mresp =  json.dumps(mresponse.body)

    searchObj = re.search(r'>Cost per Serving: (.*?)</div>',mresp)

    del missedattr['originalString']
    del missedattr['metaInformation']
    del missedattr['unitLong']
    del missedattr['unitShort']
    missedattr['price'] = str(searchObj.group(1))
    prc = re.search(r'\$(.*)',missedattr['price'])
    price = round(price + float(prc.group(1)),2)
  recipeData['missedIngredientsCost'] = price
  return recipeData

run(app, host='localhost', port=8020, debug=True)
```

Listing A.2: Python code of Personalized Coupon Engine algorithm and implementation of Foodpairing API services. The explanation of this code is in the Section 3.1 and 3.4.

```
# Usage for getting coupons: http://localhost:8050/SmartGrocer/personalizedCouponEngine?userTransactions=<filename>&clearanceGroceryList=<filename>

import sys
import json
import re
from bottle import Bottle, route, run, request, response
from Queue import Queue
from itertools import chain, combinations
from collections import defaultdict
from optparse import OptionParser
from urllib2 import Request, urlopen
import io
import subprocess
from difflib import SequenceMatcher

```

```
# To calculate similarity between two words
def similar(a, b):
    return SequenceMatcher(None, a, b).ratio()

# To determine the frequent items sets
def frequentItemsCalculation(itemSet, transactionList, minSupport, freqSet):
        _itemSet = set()
        localSet = defaultdict(int)

        for item in itemSet:
                for transaction in transactionList:
                        if item.issubset(transaction):
                freqSet[item] += 1
                localSet[item] += 1

        for item, count in localSet.items():
                support = float(count)/len(transactionList)

                if support >= minSupport:
                        _itemSet.add(item)

        return _itemSet

# Apriori algorithm
def runApriori(data, minSupport, minConfidence):
    eachTransaction = list()
    items = set()
    for transRow in data:
      transaction = frozenset(transRow)
      eachTransaction.append(transRow)
      for item in transaction:
        items.add(frozenset([item]))

    freqSet = defaultdict(int)
    frequentItemSet = dict()
    assocRules = dict()
    oneDimensionalSet = frequentItemsCalculation(items,
                                        eachTransaction,
                                        minSupport,freqSet)
    currentSet = oneDimensionalSet
    k = 2

    while(currentSet != set([])):
      frequentItemSet[k-1] = currentSet
      currentSet =set()
      for itemL in frequentItemSet[k-1]:
        for itemR in frequentItemSet[k-1]:
          if(len(itemL.union(itemR)) == k):
            currentSet.add(itemL.union(itemR))
      nextIterationSet = frequentItemsCalculation(currentSet,
                          eachTransaction,
                          minSupport,freqSet)
      currentSet = nextIterationSet
```

```
69        k+=1
70      def getSupport(item):
71          return float(freqSet[item])/len(eachTransaction)
72
73      toRetItems = []
74      for key, value in frequentItemSet.items():
75        toRetItems.extend([(tuple(item), getSupport(item))
76                   for item in value])
77
78      toRetRules = []
79      for key, value in frequentItemSet.items()[1:]:
80        for item in value:
81          _subsets = map(frozenset, [x for x in subsets(item)])
82          for element in _subsets:
83            remain = item.difference(element)
84            if len(remain) > 0:
85              confidence = getSupport(item)/getSupport(element)
86              if confidence >= minConfidence:
87                toRetRules.append(((tuple(element), tuple(remain)),
88                           confidence))
89
90      return toRetItems, toRetRules
91
92  # To generate personalized coupons and rules
93  def generateDiscountList(items, rules, grocCouponFile):
94      freqItems = [];
95      finalRules = dict()
96      print "\n---------------------------- FREQUENT ITEMS
        ----------------------------------------:"
97      for freqItem, support in sorted(items, key=lambda (item, support): support):
98          for item in freqItem:
99            freqItems.append(item)
100     finalfreqItems = list(set(freqItems))
101     finalFreqItemsString = "%2C".join(finalfreqItems)
102     print "\n---------------------------- PAIRED ITEMS
        ----------------------------------------:"
103     pairedIngredients = ingredientPairing(finalFreqItemsString, finalfreqItems)
104
105     print "\n---------------------------- RULES----------------------------------------:"
106     for rule, confidence in sorted(rules, key=lambda (rule, confidence): confidence):
107         pre, post = rule
108         pre  = json.dumps(pre)
109         post  = json.dumps(post)
110         finalRules[pre] = post
111
112
113     print "\n---------------------------- JSON DATA----------------------------------------:
        "
114     with open(grocCouponFile) as data_file:
115       groceryCoupons = json.load(data_file)
116     freqIngrCouponArr =  []
117     for eachfreqingr in finalfreqItems:
118       freqIngrCoupon = mappingCouponsWithGroceryCoupons(eachfreqingr, groceryCoupons)
```

```
119       freqIngrCouponArr.append(freqIngrCoupon)
120
121     pairIngrCouponArr =  []
122     for eachpairedingr in pairedIngredients:
123       pairIngrCoupon = mappingCouponsWithGroceryCoupons(eachpairedingr,
    groceryCoupons)
124       pairIngrCouponArr.append(freqIngrCoupon)
125
126     freqItemsJSON = dict()
127     freqItemsJSON["FrequentIngredients"] = freqIngrCouponArr
128     rulesJSON = dict()
129     rulesJSON["Rules"] = finalRules
130     pairedJSON =  dict()
131     pairedJSON["PairedIngredients"] = pairIngrCouponArr
132     jsonData = dict()
133     jsonData["Data"] =list()
134     jsonData["Data"].append(dict())
135     jsonData["Data"][0].update(freqItemsJSON)
136     jsonData["Data"][0].update(rulesJSON)
137     jsonData["Data"][0].update(pairedJSON)
138     return jsonData
139
140 # To map frequent and paired ingredients with Clearance Grocery List
141 def mappingCouponsWithGroceryCoupons(ingr, coups):
142   for IngrCoupons in coupons["Data"]:
143     for eachCoupon in IngrCoupons["IngredientsCoupons"]:
144       if similar(eachCoupon["name"],eachfreqingr) >0.7:
145         ingrCoup = dict()
146         ingrCoup["name"] = eachCoupon["name"]
147         ingrCoup["discount"] = eachCoupon["discount"]
148   return ingrCoup
149
150 # To find the pairing ingredients list from the frequently purchased ingredients list
151 def ingredientPairing(ingredients, freqItems):
152   headers = {
153     'X-Application-ID': 'APP_ID',
154     'X-Application-Key': 'API_KEY'
155   }
156   ingredients = re.sub(r'\s','%20',ingredients)
157   # calling Foodpairng Search Ingredients API request to determine the ids
158   url = 'https://api.foodpairing.com/ingredients?q='+ingredients
159   request = Request(url, headers=headers)
160   response_body = urlopen(request).read()
161   response_body = json.loads(response_body)
162   for eachIngredient in response_body:
163     del eachIngredient['name']
164     del eachIngredient['preparation']
165     del eachIngredient['_links']
166     del eachIngredient['description']
167     del eachIngredient['_meta']
168   frequentItemsId = []
169   for freqIngr in freqItems:
170     id  = subprocess.check_output([sys.executable,"posTag_ingrMatch.py",freqIngr, str
```

```
    (response_body)])
171     if(id != ''):
172       frequentItemsId.append(id.rstrip())
173 
174   freqIdString = ",".join(frequentItemsId)
175   # calling Foodpairng Pairing Ingredients API request to determine the paired
        ingredients list
176   url = 'https://api.foodpairing.com/ingredients/'+freqIdString+'/pairings'
177   request = Request(url, headers=headers)
178   pairingResponse = urlopen(request).read()
179   pairingResponse = json.loads(pairingResponse)
180   finalpairedIngredients = []
181   for eachPairIngr in pairingResponse:
182     if(eachPairIngr['matches']['all']['abs'] >0.8):
183       finalpairedIngredients.append(eachPairIngr['_links']['ingredient']['product'])
184 
185   return finalpairedIngredients
186 
187 def subsets(arr):
188     return chain(*[combinations(arr, i + 1) for i, a in enumerate(arr)])
189 
190 def fileData(filename):
191   file = open(filename, 'r')
192   wholeData = []
193   for line in file:
194     line = line.strip().rstrip(',')
195     row = line.split(',')
196     wholeData.append(row);
197   return wholeData
198 
199 app = Bottle()
200 
201 @app.hook('after_request')
202 def enable_cors():
203   """
204   You need to add some headers to each request.
205   Don't use the wildcard '*' for Access-Control-Allow-Origin in production.
206   """
207   response.headers['Access-Control-Allow-Origin'] = '*'
208   response.headers['Access-Control-Allow-Methods'] = 'PUT, GET, POST, DELETE, OPTIONS
        '
209   response.headers['Access-Control-Allow-Headers'] = 'Origin, Accept, Content-Type, X
        -Requested-With, X-CSRF-Token'
210 
211 @app.route('/SmartGrocer/<name>', method=['GET'])
212 def recData(name):
213   if name == "personalizedCouponEngine":
214     inputTransFile = fileData(request.query.userTransactions)
215     clearGrocFile = request.query.clearanceGroceryList
216     # setting confidence value.
217     confidence = 0.6
218     # setting support value between 0.1 and 0.2
219     support = 0.15
```

```
220     # applying Apriori algorithm
221     items, rules = runApriori(inputTransFile, support, confidence)
222     # generate Discount Ingredients list
223     discountItems = generateDiscountList(items, rules, clearGrocFile)
224     return dict(discountItems)
225
226
227 run(app, host='localhost', port=8050, debug=True)
```

Listing A.3: Python code of POS Tagging Mean Probability algorithm. The explanation of this code is in the Section 3.3

```
1  from nltk import pos_tag, word_tokenize
2  import sys
3  import json
4  import urllib2
5  import requests
6  import xml.etree.ElementTree as ET
7  from xml.etree.ElementTree import XML, fromstring, tostring
8  import difflib
9
10 # To determine the grocery ingredients list based on recipe ingredient nouns.
11 def grocAPIResponse(tagItemNoun):
12   apicall = "http://localhost:8070/UStopGroceryStore?SearchText="+tagItemNoun
13   data = requests.get(apicall)
14   xmlData = data.content
15   tree = ET.fromstring(xmlData)
16   root = {}
17   items = []
18   for child in tree:
19     root[child.tag.split("}")[1]] = child.text
20     item =  root["string"]
21     productName = item.split("-")[0]
22     items.append(productName)
23
24   return items
25
26 # To apply POS tagging on ingredient name
27 def posTagResponse(ingredient):
28   posResult = pos_tag(word_tokenize(ingredient))
29   posTagJSON = {}
30   noun = []
31   verb = []
32   adjective = []
33   for tagTuple in posResult:
34     if(tagTuple[1] == "NN" or tagTuple[1] == "NNS" or tagTuple[1] == "NNP" or
     tagTuple[1] == "NNPS" ):
35       noun.append(tagTuple[0].lower())
36     elif(tagTuple[1] == "VB" or tagTuple[1] == "VBD" or tagTuple[1] == "VBG" or
     tagTuple[1] == "VBN"  or tagTuple[1] == "VBP"  or tagTuple[1] == "VBZ" ):
37       verb.append(tagTuple[0].lower())
38     elif(tagTuple[1] == "JJ" or tagTuple[1] == "JJR" or tagTuple[1] == "JJS" ):
```

```
39        adjective.append(tagTuple[0].lower())
40    if(len(noun)>=1):
41      posTagJSON["Noun"] = noun
42    if(len(verb)>=1):
43      posTagJSON["Verb"] = verb
44    if(len(adjective)>=1):
45      posTagJSON["Adjective"] = adjective
46
47    return posTagJSON
48
49  # To calculate the probability between recipe and grocery ingredient in terms of noun
      , verb, and adjective probability
50  def probabilityCalculation(tag, recipeJSON, groceryJSON):
51    counter = 0
52    sum = 0.0
53    prob  = 1.0
54    if(tag in recipeJSON and tag in groceryJSON):
55      for tagR in recipeJSON[tag]:
56        for tagG in groceryJSON[tag]:
57          p = difflib.SequenceMatcher(None,tagR,tagG).ratio()
58          if(p>0.7):
59            sum += p
60          counter+= 1
61      if(counter>0):
62        prob = sum /counter
63
64    return prob
65
66  if __name__ == "__main__":
67    recipeIngredient = "juice of lime"
68    # calling function to get set of nouns, verbs, and adjectives
69    recipeIngredientJSON  = posTagResponse(recipeIngredient)
70    groceryIngredients = []
71    grocAPIResponseForRecipeIngr = grocAPIResponse(recipeIngredient)
72    if(len(grocAPIResponseForRecipeIngr)>=1):
73      groceryIngredients = grocAPIResponseForRecipeIngr
74    else:
75      for noun in recipeIngredientJSON["Noun"]:
76        groceryIngredients += grocAPIResponse(noun)
77
78    sum  = 0.0
79    counter = 0
80    pairs = {}
81    # setting max mean probability to 0.7
82    maxProbability  = 0.7
83    maxPair = ()
84    nounProbability = 0
85    verbProbability = 0
86    adjProbability =  0
87    for groceryIngredient in groceryIngredients:
88      # calling function to get set of nouns, verbs, and adjectives
89      grocJSON = posTagResponse(groceryIngredient)
90
```

```
91      # determining the noun, verb, and adjective probability
92      nounProbability = probabilityCalculation("Noun", recipeIngredientJSON, grocJSON)
93      verbProbability = probabilityCalculation("Verb", recipeIngredientJSON, grocJSON)
94      adjProbability = probabilityCalculation("Adjective", recipeIngredientJSON,
        grocJSON)
95
96      totalProbability =  (nounProbability + verbProbability + adjProbability)/3
97      matchingPair = (recipeIngredient,groceryIngredient)
98      pairs["matchingPair"] = matchingPair
99      pairs["probability"] = nounProbability
100
101     if(totalProbability> maxProbability):
102       maxProbability = totalProbability
103       maxPair = matchingPair
104
105   print maxProbability
106   print maxPair
107
```