COOKBETTER ver 2 : A Bot for Personalized Recipe Recommendation

[Report 2c - Team 'O' enhanced Team 'E's work]

Shrikanth N C
NC State University
CS department, NC 27606
snaraya7@ncsu.edu

Karthik M S
NC State University
ECE department, NC 27606
kmedidi@ncsu.edu

Kashyap S
NC State University
ECE department, NC 27606
ksivasu@ncsu.edu

Charan Ram V C
NC State University
ECE department, NC 27606
cvellai@ncsu.edu

Ragavi Kalaignani
NC State University
CS department, NC 27606
rkalaig@ncsu.edu

ABSTRACT

The goal of this project is to improve *Cookbetter bot* by introducing new features and bolstering existing features to provide better usability and applicability of the system. Firstly, we discern existing problems of the current version by analyzing the evaluation report to identify the points where the application required betterment. Secondly, we outline each solution with their use cases that are built to achieve the aforementioned goal. Then the report subsequently discusses all the use cases in detail which we believe have enhanced the functional flexibility of the system, thereby increasing the overall efficiency and ultimately offering high-grade user experience. Lastly, the report outlines the development process to be adopted with the timeline that was required for completion and presents the evaluation results of the project.

Keywords

Recommendation System, Slackbot, Image, Text processing, Visualization, Feedback

1. WHAT WAS DONE BEFORE?

1.1 Overview

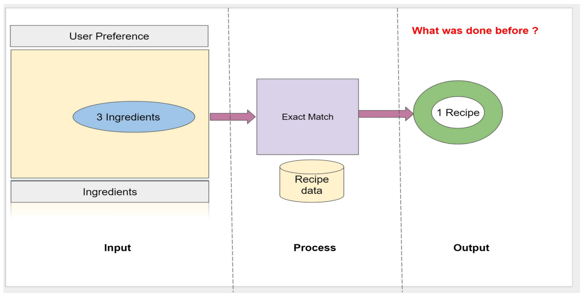


Figure 1: What was done before?

The existing system 1, Cook Better, is a Slack chat bot for personalized recipe recommendation. It allows the user to configure their dietary and health restrictions in the system. Users can search for recipes based on ingredients and few other criteria. Recipe data from the website Epicurious is used. Based on the ingredients and criteria chosen at the time of searching and also based on each user's preconfigured dietary and health restrictions, recipes are selected from the database. A list of recipe links is displayed to the user, and upon clicking these links the user is redirected to the Epicurious website to get the exact instructions for cooking. The chat bot currently supports four slash commands - /personalize, /searchrecipes, /surpriseme and /cookbetterhelp.

1.2 Evaluation

Evaluation of the existing system has been carried out by allowing users to interact with the chat bot in a Slack workspace. After using the chat bot, users were asked to fill an online questionnaire. The questions were chosen to validate the following four metrics:

- Reliability
- Scalability
- Cooperativity
- User Satisfaction

1.3 Results

The results of the evaluation that was conducted clearly indicated that the evaluators found the chat bot to be both helpful and reliable. The users also indicated that they would prefer using this chat bot over traditional websites for recipe recommendation.

1.4 Problems

The evaluation results revealed three major problems with the system. Most of the suggestions that were given to change the existing system were broadly related to these problems:

1. **Input**: There were multiple suggestions to use text boxes, checkboxes or radio buttons for selecting ingredients, rather than using drop-down select boxes. Many suggestions mentioned that a more intuitive and interactive user interface would be helpful.

2. **Output**: Some evaluators had these problems with the output - redirecting to a website instead of giving more information within the bot, ending the search abruptly after displaying the results without having an option to continue the search or get more details.

3. **Recommendation**: Some evaluators suggested the incorporation of machine learning algorithms to improve the quality of results that are displayed to the user.

1.5 Learnings

Upon studying the existing system and examining the evaluation results, the following are our major takeaways:

- We need to make the method of inputting ingredients more intuitive and easy-to-use.
- We need to enhance the output that is displayed to the user and provide more information.
- We need to develop an intelligent recommendation module to search for recipes.

2. WHAT WE DID?

This section will give a brief description of how we have enhanced the older version of Cookbetter by incorporating the lessons learned from the previous evaluation results of Cookbetter version 1.

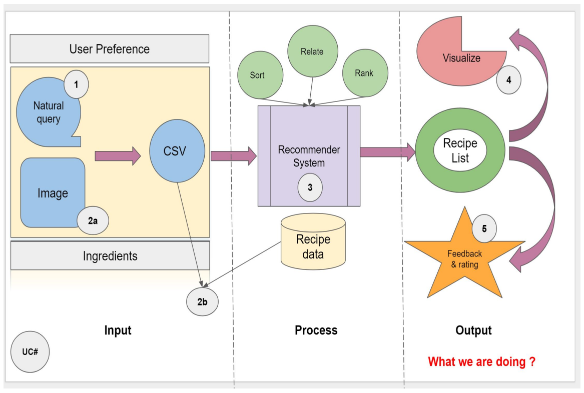


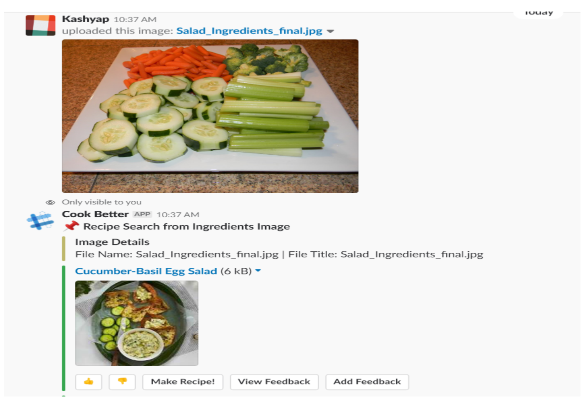
Figure 2: What we did

The above Figure 2 shows the different use cases that we have built for version 2 of Cookbetter. A detailed description of each use case is given in the following sections.

2.1 Image to recipe

The Image to Recipe feature that we have developed is probably the coolest and simplest way to interact with the application. The feature allows a user to simply take a picture of all the ingredients that he wants to cook a dish with and upload the picture to the application. The Figure 3 shows a sample output of the image search feature.

The application identifies each ingredient from the picture and returns possible dishes from the database that can be cooked with the best mix of ingredients. We have used an Image Recognition recipe that seamlessly does the job of identifying individual ingredients from the uploaded image.


Kashyap 10:37 AM
uploaded this image: Salad_Ingredients_final.jpg

Only visible to you
Cook Better APP 10:37 AM
📌 Recipe Search from Ingredients Image
Image Details
File Name: Salad_Ingredients_final.jpg | File Title: Salad_Ingredients_final.jpg
Cucumber-Basil Egg Salad (6 kB)

Figure 3: Image Search

2.2 Natural Query

Another way by which we have tried to make the input a lot more intuitive is by using a free-form text input for searching for dishes instead of using drop downs to select ingredients. The user was earlier selecting ingredients and recipe type from drop downs to search for recipes as shown in the Figure 4.

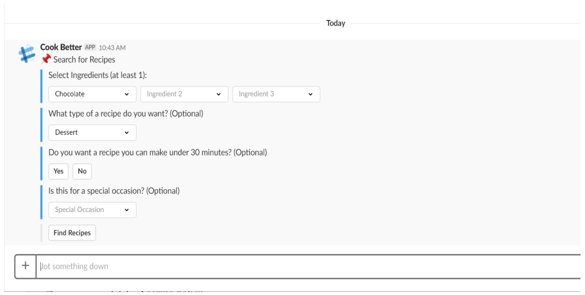

Cook Better APP 10:43 AM
📌 Search for Recipes
Select Ingredients (at least 1):
Chocolate | Ingredient 2 | Ingredient 3
What type of a recipe do you want? (Optional)
Dessert
Do you want a recipe you can make under 30 minutes? (Optional)
Yes | No
Is this for a special occasion? (Optional)
Special Occasion

Figure 4: Searching Recipes:The Old Way

So, in the new version, the user can just give the name of the dish as an input or even directly give the corresponding ingredients separated by a comma as shown in the figures below. This gives user higher flexibility to search for a recipe. The following figures will highlight this feature better.

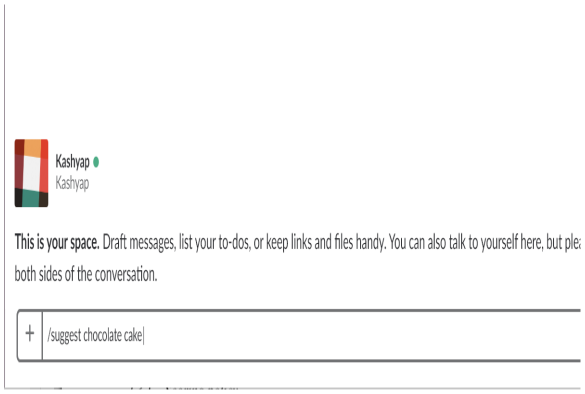
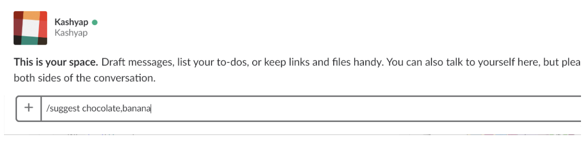

Kashyap
Kashyap
This is your space. Draft messages, list your to-dos, or keep links and files handy. You can also talk to yourself here, but ple
both sides of the conversation.
/suggest chocolate cake

Figure 5: Searching Recipes: The New Way

We can see as shown in Figure 5 that search is done by directly searching for the name of the recipe and in the Figure 6 below ingredients are given as input separated by commas.

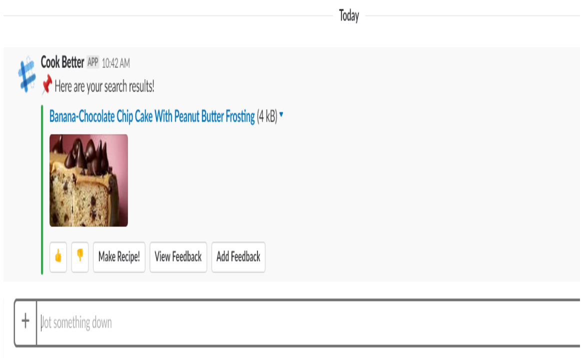

Kashyap
Kashyap

This is your space. Draft messages, list your to-dos, or keep links and files handy. You can also talk to yourself here, but plea
both sides of the conversation.

/suggest chocolate,banana

Figure 6: Searching Recipes: The New Way

The Figure 7 shows the dish returned as output for both the inputs. We can see that this is a more intuitive and a more practical way of searching for a dish rather than choosing ingredients from drop downs.


Today

Cook Better APP 10:42 AM
Here are your search results!
Banana-Chocolate Chip Cake With Peanut Butter Frosting (4 kB)

Figure 7: Searching Recipes: The New Way (Result)

2.3 Recommender System

The recommender systems accept inputs from various channels as discussed earlier. For recommending the system also considers self-explanatory configuration parameters as shown in image 8. In a nutshell, the recommender system ranks the recipes based on a score which is detailed in the next section.

2.3.1 *Scoring policy*

As shown in the figure 8 given a set of ingredients the user is recommended the closest recipe in a cost-effective way. In other words, the scoring minimizes the need for the user to purchase additional ingredients which certain recipes may require. Missing and extra ingredients contribute to the add-on cost for the user. Certain essential ingredients may be missing in a recipe which the user requested. In some situations the user's ingredients may be a subset of the recipe's ingredients, however, there will be certain additional ingredients in the recipe. Both of these cases are usually unavoidable hence we try to deduct the score accordingly keeping the matching ingredient weight significantly higher.

2.3.2 *Algorithm*

The algorithm operates on the scoring policy discussed earlier. Each recipe is assigned a score relative to the ingredient list received from the user. The recipes are ranked

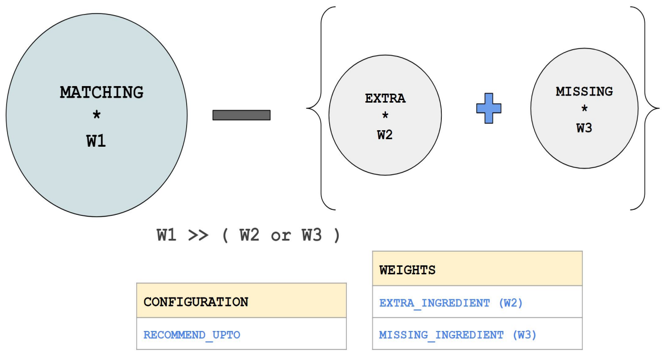


Figure 8: Score Policy

with the assigned score. Simply the top recipes with highest scores are presented to the user. In the figure 9 X refers to the score based on the scoring policy and Y refers to the configurable value to recommend upto in the figure 8

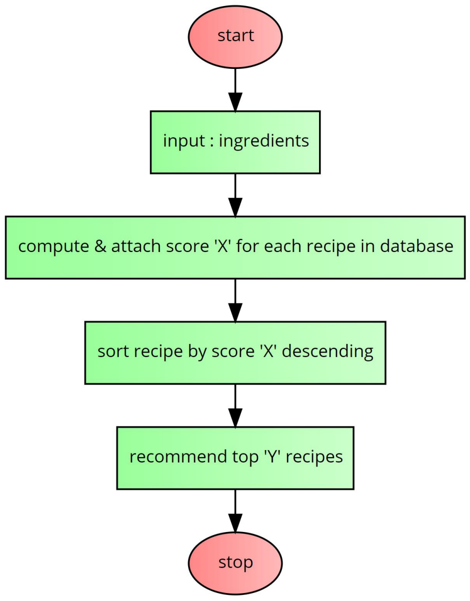

start
input : ingredients
compute & attach score 'X' for each recipe in database
sort recipe by score 'X' descending
recommend top 'Y' recipes
stop

Figure 9: Flowchart

2.4 Visualization

The visualizations feature was introduced to give a pictorial representation of the recipes that we have in the database with the specific pattern of usage of our application taken into consideration. Currently we have a simple chart representing the number of likes of a type of a dish given the limitation in database that we had. We have classified the types of dish as vegetarian, vegan and other (dishes with meat).

The chart that is returned from the application is shown as below. Currently there are no likes for Vegan Dishes in our database.

So, the stats gives us a particular pattern of usage of our application. The same visualization technique with a better

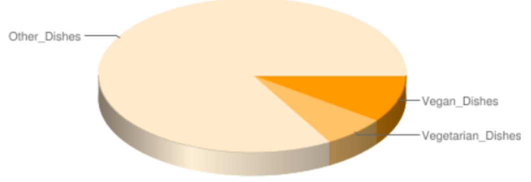


Figure 10: Recipe Stat

database be extended to find the most sort after cuisine in a particular location or say finding the dietary preference of people. All these data if collected with privacy into consideration, would open an avenue in data analytics for food preference of people, which could in turn be useful for food retailers to serve their customers better.

3. WHY WE DID THIS?

The simple and versatile manner in which the project cookbetter addressed the problem statement in question was one of the prime factors that caught our attention. Their idea of using Slack as a base for implementing the user interface was well thought of while keeping the time constraints in mind. Although the application was well built it simultaneously presented a wide scope of improvement which ultimately led us to select this project for our second phase. The requirements for the enhanced version of cookbetter application were gathered from the previous version's evaluation which enabled us to identify certain key aspects of the application that warranted improvements. By doing so, we acquired inputs from the end users of the application and could develop use cases that incorporated those feedback. The tasks associated with developing the use cases were derived and assigned to all the team members such that everybody could work parallelly. The Protoyping model of software development was adopted considering its simplicity and the time span of the project. Thus, the focus was initially on the Image to recipe feature which seemed to be the most difficult to conceive among all other use cases. Test cases were also written and the developing code was periodically tested through unit and system tests to conform with the Test Driven Methodology. In this way, we had hoped to fix modest errors within our code early on during the development phase which could otherwise prove to be time-consuming in the end and hence leave little time for the evaluation phase. Ultimately, through our efforts, we sought to develop a framework through which the primary objective could be achieved in compliance with accepted software development principles

4. INFRASTRUCTURE

We are enhancing the existing system without affecting the current architecture. We will be adding our enhancements to the REST service running on the server. The architectural design is shown in Figure 11.

When the user interacts with the chat bot, Slack sends a POST request to our server running on AWS [1]. The Spring application [5] queries the MySQL database [4] and constructs a JSON response. The response is sent to Slack

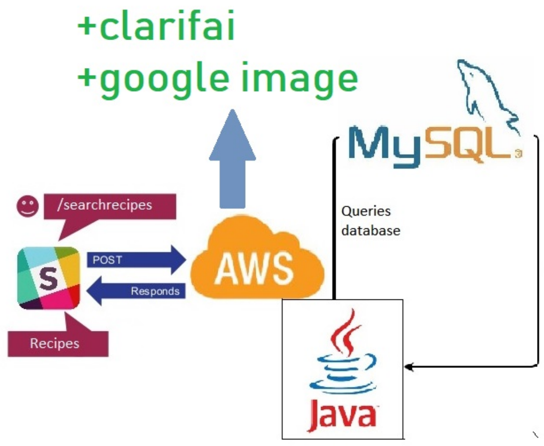


Figure 11: Architecture

and displayed to the user. Major new additions include just calls to clarifai a computer vision based api and goole images domain.

5. TIME-LINE

We recognized and allocated the tasks among ourselves and followed a modularized approach to software development to incorporate parallelism as far as possible. By doing so we had hoped to find and fix errors faster and complete development such that we had sufficient time for a proper validation phase. The actual time-line of activities for this project shown in Figure 12 will substantiate the same.

6. EFFORT ESTIMATION

6.1 Expected

We have used Use Case Points [2] to arrive at an estimate for the effort required for the implementation of the feature enhancements discussed in this document. Figure 13 shows the results of our effort estimation. In order to complete our project, we have come up with an estimate of 187 man hours. The Technical Complexity of the project, as well as several environmental factors, have been considered while determining the effort required. The motivation within the team, familiarity with the project, programming experience of the team members and the capability of the lead analyst were some of the environmental factors that played a role in the estimation process. The reusability of the existing code base and the use of third-party software and libraries (Actors) were also taken into account to provide a more accurate result. We understand that this is not an ideal estimate. However, this activity helped us to dissect and view the factors that could affect the delivery of our application.

6.2 Actual

The actual effort required for the project was lesser than the estimate. We didn't accurately record or maintain a time sheet. However, if we have to roughly estimate each of us has spent an hour per day for 35 (Mar: 12 + Apr: 23) days. That is a total of *175* hours (+ or - 10 hours). This includes

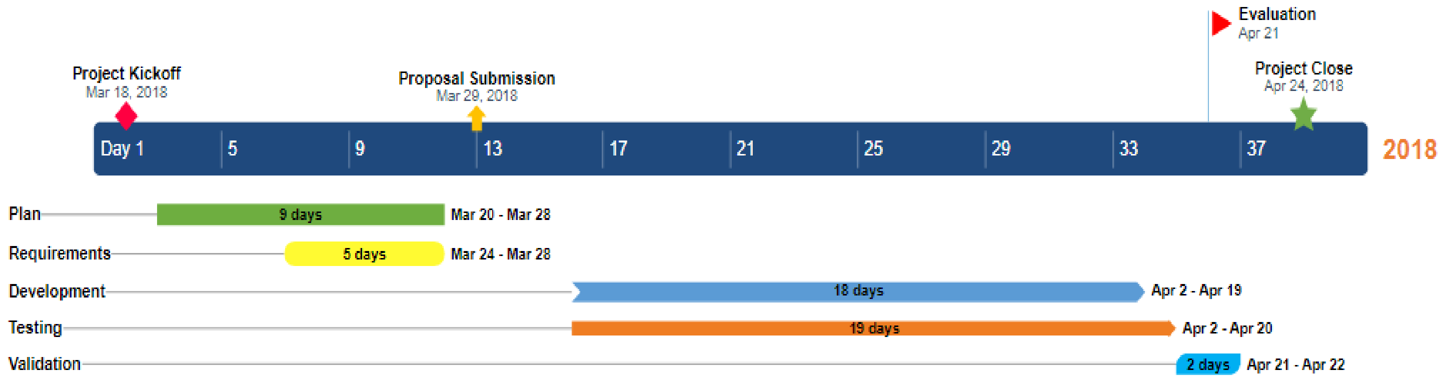
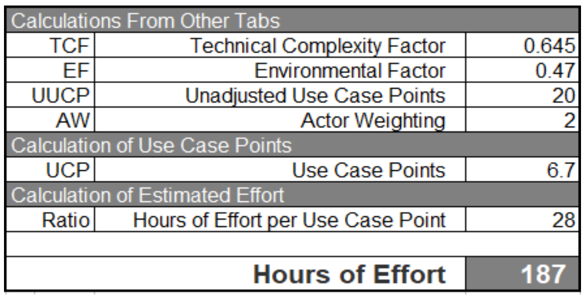


Figure 12: Project Time-Line

all phases of the software development life cycle and other peripheral activities such as this document preparation. The actual effort approximately matches the result of the effort estimation model that was put forth in the initial design document.



Calculations From Other Tabs		
TCF	Technical Complexity Factor	0.645
EF	Environmental Factor	0.47
UUCP	Unadjusted Use Case Points	20
AW	Actor Weighting	2
Calculation of Use Case Points		
UCP	Use Case Points	6.7
Calculation of Estimated Effort		
Ratio	Hours of Effort per Use Case Point	28
	Hours of Effort	187

Figure 13: Effort Estimation results

7. SOFTWARE DEVELOPMENT LIFE CYCLE

A software development life cycle is aimed at enabling rapid development of software with minimization of errors through substantial planning and management. It is crucial to adopt a software development process which accurately describes the aspects of the project in hand considering the requirements specified and resources available. Bearing these factors in mind, a Prototyping model of development has been chosen. Further, this model of development provides a means to deliver results early in the timeline of the project. We also adopted TDD as a primary model for development. TDD also helped us to streamline our approach enabling us to adhere to the requirements of the project. The management and support tasks for the project was accomplished through github primarily and the development was done on IntelliJ Idea IDE for coding, testing and compilation of the code.

7.1 Plan

This was the initial stage of our development process where we found the scope of the problem and possible solutions for the problem. First, the problem was identified as increasing the user experience of the application and need of sophisticating the application. The scope of the problems was analyzed with the timeline and resources in hand and the five use cases discussed in section 2 were proposed as a result.

7.2 Requirements

In this section, we will see in detail about how we collected the requirements for improving the existing project. The inputs were majorly from previous evaluation report which included suggestion from peers. From the inputs that were gathered, the 5 use cases that were discussed in section 2 were decided to be worked on. All the use cases which are proposed in Section 2 in one way or the other tries to better the user experience of the application and aims at increasing functional flexibility of the application.

7.3 Design & Implementation

The method of implementation that we have thought of is to divide the development of the new use cases among the team members and each team member will be responsible for writing a test case for another use case that the member is not working on. This would make each team member gain exposure to both development and testing side of the development process and would also give functional knowledge of two use cases of the application. As stated above, we had followed the TDD methodology, which helped us to find critical errors at an early stage of development and enabled us to rectify them rapidly.

7.3.1 Class Diagram

Refer to the class diagram in figure 14 all the classes revolve around the central class RecommenderSystem that takes a set of ingredients to recommend a stack of recipes. Some users may prefer to see fewer recommendations and some a lot. The same can be configured in the RecommenderConfiguration class by setting the RECOMMEND_UPTO variable. The recommender system is designed to add recipes to the stack based on the values set in this variable. As discussed earlier we used a simple weight based equation that takes account of extra and missing ingredients. The weight of the same can be configured here in the RecommenderConfiguration class. Recipe, Ingredient as obviously the dependent classes to the RecommenderSystem. We discussed about attaching score to each recipe and sorting them later in the flowchart. The RecipeWrapper is needed to capture such metadata for the RecommenderSystem to operate on.

GoogleImageFetcher is used to search and crawl recipe images in the images.google.com portal. The util classes like DatabaseUtil, FeedbackUtil and Util are self explanatory.

7.4 Test

It is highly essential that any software built must be subject to rigorous testing before making it available for usage. This testing must not only aim at maintaining a high quality of code but also provide a mechanism to ensure that the user requirements have been satisfied accurately. Following a test-driven methodology has thus been adopted for our project. This allows maintaining simplicity, modularity, and flexibility in the code base while also providing a built-in process of evaluation against the design. Our objective would be to build test cases to each of the requirements corresponding to the use cases presented in section 2. This further enables us to obtain the perspective of an end user as TDD makes the developer focus on the requirements before writing the code. Following the construction of these test cases, the actual software coding is done to pass the test cases. Intuitively, this may result in failure, but this provides crucial insight into the short-comings of the current code. Next, the code is improved with an aim of barely passing the existing test cases to satisfy the bare-bone design requirements. The written code is then validated against all the test cases and upon success, the code is deemed to meet the test requirements.

7.5 Deploy & Maintenance

As we know, any web-based application requires very little effort on part of the user for the installation. The architecture of our software application uses a web archive file that is deployed on a remote server hosted on an Amazon Web Services Elastic Computing instance. The user interaction with our application is through slash command requests to the Slack interface which invokes the Slack API and then this request is redirected to an URL hosted on our server. A Kaggle database is maintained at this server which acts as the knowledge base of recipes for processing. This simple design has allowed us to improve upon processing times and provides a performance boost by removing redundant interactions between different modules of the software.

8. CHALLENGES

Apart from learning and adapting to new technology stack, we faced two application related challenges.

8.1 Plurality

An impedance that tampered with smooth automation. *if apple = apples unfortunately cherry = cherries* We could have used NLP libraries but we thought to handle this way because the ingredients subset in the database was close to 600 as opposed to all the ingredients in the world.

```
void match(longer, shorter, start, end){

if (longer.contains(shorter)){
   score = Math.max(shorter.length(), score
    );
}
else if (start < end && end < shorter.
    length()) {
// recursive match
match(longer, shorter.substring(start, end)
    , start, (end+1));
match(longer, shorter.substring(start, end)
    , (start+1), end);

    }
}
```

Listing 1: Ingredient matching

This algorithm shown in Listing 2 counts the longest substring between two strings. Here longer and shorter refers to the ingredient names. start and end refers to the string index.

8.2 Performance

Though an image predominantly have ingredients it may also have other data that image recognition api's detect. Now each of these non-ingredients are unfortunately compared with the 600 fixed ingredients. A simple statically declared Map (HashMap in java) has proven to be very effective in improving the performance by caching an ingredient request. We also persist this map by serializing the same and deserializing on a fresh load.

```
 String matchWithExisting(
    receivedIngredient){
     matchingIngredient = ingredientCache.
    get(receivedIngredient);

     if (matchingIngredient == null){

          ingredientCache.put(
    receivedIngredient, find(
    receivedIngredient));
     }
     return ingredientCache.get(
    receivedIngredient);
}
```

Listing 2: Ingredient request cache

9. PROJECT MANAGEMENT

To avoid ambiguity and maintain clarity we divided large tasks into small chunks with short deadlines. We created a Google drive folder dedicated to this project and all the assets pertaining to this project went into that folder. As suggested by the instructors we maintained different branches in git for safe and parallel development.

Component	Owner(s)
Recommender, Plurality, Caching	Shrikanth N C
Natural Query, Visualisation	Kashyap S
Review and Rating system	Karthik M S
Database Management, Testing	Charan Ram V C
Image to Recipe, Infra Management	Ragavi

Table 1: Divide & Conquer

In situations when members couldn't collaborate in person to address certain issues, we used Team viewer software to share our screens to rectify the same. Git merge can be arduous sometimes, especially for newbies hence we used beyond compare tool for all local merges.

Asset/Tool	About
Task Management spreadsheet	Manage tasks
Team viewer	Enable remote working
Beyond compare	Local code comparison
Git kanban and issues	milestone management
Google drive	Collaboration

Table 2: Internal Evaluation

9.1 Testing

We focused predominantly on system testing and acceptance testing since we orchestrated few third party api's hosted in a cloud infrastructure. Having said that few unit test cases are found in the src/test/java folder.

10. EVALUATION

The evaluation process for this project focuses on **quantitatively** characterizing the possible benefit due to the feature upgrades that have been made to the existing software. A primary emphasis on this concern has been placed while designing the evaluation for the software.

10.1 Procedure

A survey has been selected as the evaluation tool for this project and the survey points have been made specific to center on the feature upgrades. Extra care was taken to avoid skewing of participant opinion through objective measurements of time taken and recording the user's honest feedback. The following sections are some of the significant survey points.

10.2 Task Completion

A list of tasks were asked to be completed by the user and the time taken to complete each of these tasks and any difficulty faced in completing a task were recorded. The tasks were designed to simulate real-world user quandaries and will capture the nuances of user requirements in increasing proportion.

The survey participant's feedback on whether the new features contribute to the completeness of the solution with respect to the initial problem statement that the software seeks to tackle has been recorded. Also, the evaluation will determine if the goal of enabling user engagement with the software has been met. Further, the intuitive recipe recommendation system that is part of the new release of the software targets the accuracy with which the user's request is accomplished. The user experience difference due to this feature from the previous version of the software has been recorded.

10.3 Usability

As part of the new release, the forms of input to the software were increased. We sought to record the user's preference to one or more types of input. In this regard, users were asked to provide their feedback on using these features.

10.4 Results of cookbetter version 2.0

The **/suggest** feature in the second version of cookbetter aimed to bring in free-form text as a form of input taking into account the fact that Slack in itself was a free-form text based application. Our hope was to extend the ease of interaction that Slack promotes to our software. The **/searchrecipes** command reproduces the version 1.0's signature form of input while the **/imagesearch** tried to test the adaptability to a new form of input in the form of a picture. The following figures 15, 16, 17 time taken to complete a unit task using the corresponding input styles.

The remaining survey points pertained to the software feel and sought to measure the adaptability of these features. Figure 18 is a measurement of the user's preference for in-app directions (version 2.0) versus a bland hyperlink output (version 1.0)

The figures 19 and 20 can be used to compare user's comfort in using the features of /suggest and /imagesearch respectively. Users were asked objective queries of preference to interact with the software with either of these methods.

10.5 Inferences

The following inferences were made from the results of the evaluation process.

10.5.1 Positive

The goal of the evaluation process was to assess not only if the software serves its primary objective but also how this objective has been achieved. It is this latter portion of our inquiry that was the primary focus of the evaluation process. The objective measures of time taken to complete a task in the software helped us to ascertain if the initial goal was met. From the evaluation results, we can conclude that the actual time taken by the users was roughly the same as the expected time. The /suggest feature which involved typing cost the user the least time with more than 75% of the participants completing the task in five seconds or less.

On the other hand, /searchrecipes form based input style was expected to take a little longer and correspondingly, 78.6% particpants completing the task in fifteen seconds or less. Thus we conclude that the design of our version has achieved our expectations for a solution to the problem statement.

10.5.2 Negative

However, From the results of the evaluation of the software feel, our expectations quite did not match the actual users' requirements. We had expected that most users would prefer in-app directions over a bland hyperlink but it was found from our results that there was a variety of preference for either style and each accounted for a major percentage of the participants.

The /imagesearch feature was a cause of concern because of the significant challenges faced in proper image recognition and integration into Slack. Our postulate was confirmed through our survey findings wherein more than 85% of the participants took fifteen seconds or longer to complete a task through this form of interaction.

10.5.3 Chits

The chits collected from our class students who evaluated our project are as follows : REE-MZC-GDJ-LNS-NUU-RID-VIZ-HAJ-YJJ-WBM-LZZ-VJW-UWY-UKC.

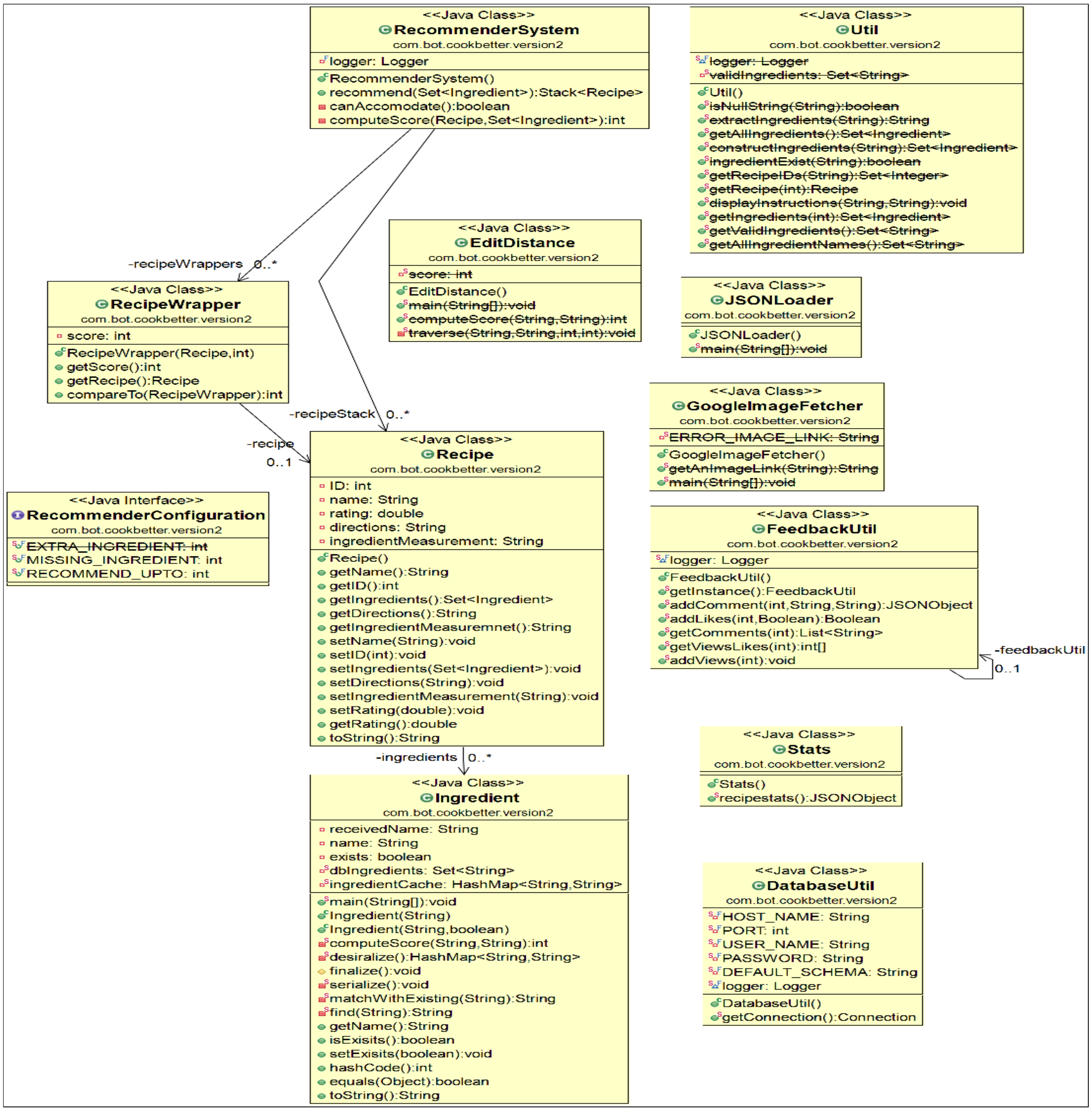

<<Java Class>>
RecommenderSystem
com.bot.cookbetter.version2
logger: Logger
RecommenderSystem()
recommend(Set<Ingredient>):Stack<Recipe>
canAccomodate():boolean
computeScore(Recipe,Set<Ingredient>):int
<<Java Class>>
Util
com.bot.cookbetter.version2
~~logger: Logger~~
~~validIngredients: Set<String>~~
Util()
~~isNullString(String):boolean~~
~~extractIngredients(String):String~~
~~getAllIngredients():Set<Ingredient>~~
~~constructIngredients(String):Set<Ingredient>~~
~~ingredientExist(String):boolean~~
~~getRecipeIDs(String):Set<Integer>~~
~~getRecipe(int):Recipe~~
~~displayInstructions(String,String):void~~
~~getIngredients(int):Set<Ingredient>~~
~~getValidIngredients():Set<String>~~
~~getAllIngredientNames():Set<String>~~
<<Java Class>>
EditDistance
com.bot.cookbetter.version2
~~score: int~~
EditDistance()
~~main(String[]):void~~
~~computeScore(String,String):int~~
~~traverse(String,String,int,int):void~~
-recipeWrappers
0..*
<<Java Class>>
RecipeWrapper
com.bot.cookbetter.version2
score: int
RecipeWrapper(Recipe,int)
getScore():int
getRecipe():Recipe
compareTo(RecipeWrapper):int
<<Java Class>>
JSONLoader
com.bot.cookbetter.version2
JSONLoader()
~~main(String[]):void~~
<<Java Class>>
GoogleImageFetcher
com.bot.cookbetter.version2
~~ERROR_IMAGE_LINK: String~~
GoogleImageFetcher()
~~getAnImageLink(String):String~~
~~main(String[]):void~~
-recipeStack
0..*
-recipe
0..1
<<Java Class>>
Recipe
com.bot.cookbetter.version2
ID: int
name: String
rating: double
directions: String
ingredientMeasurement: String
Recipe()
getName():String
getID():int
getIngredients():Set<Ingredient>
getDirections():String
getIngredientMeasuremnet():String
setName(String):void
setID(int):void
setIngredients(Set<Ingredient>):void
setDirections(String):void
setIngredientMeasurement(String):void
setRating(double):void
getRating():double
toString():String
<<Java Interface>>
RecommenderConfiguration
com.bot.cookbetter.version2
~~EXTRA_INGREDIENT: int~~
MISSING_INGREDIENT: int
RECOMMEND_UPTO: int
<<Java Class>>
FeedbackUtil
com.bot.cookbetter.version2
logger: Logger
FeedbackUtil()
getInstance():FeedbackUtil
addComment(int,String,String):JSONObject
addLikes(int,Boolean):Boolean
getComments(int):List<String>
getViewsLikes(int):int[]
addViews(int):void
-feedbackUtil
0..1
-ingredients
0..*
<<Java Class>>
Stats
com.bot.cookbetter.version2
Stats()
recipestats():JSONObject
<<Java Class>>
Ingredient
com.bot.cookbetter.version2
receivedName: String
name: String
exists: boolean
dbIngredients: Set<String>
ingredientCache: HashMap<String,String>
main(String[]):void
Ingredient(String)
Ingredient(String,boolean)
computeScore(String,String):int
desiralize():HashMap<String,String>
finalize():void
serialize():void
matchWithExisting(String):String
find(String):String
getName():String
isExisits():boolean
setExisits(boolean):void
hashCode():int
equals(Object):boolean
toString():String
<<Java Class>>
DatabaseUtil
com.bot.cookbetter.version2
HOST_NAME: String
PORT: int
USER_NAME: String
PASSWORD: String
DEFAULT_SCHEMA: String
logger: Logger
DatabaseUtil()
getConnection():Connection

Figure 14: Class diagram

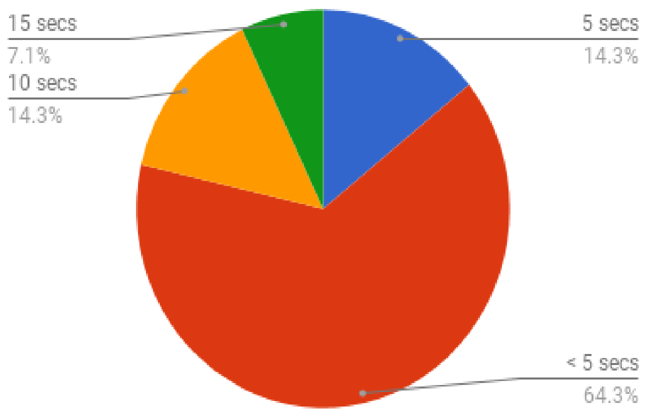


Figure 15: Time taken per task using '/suggest'

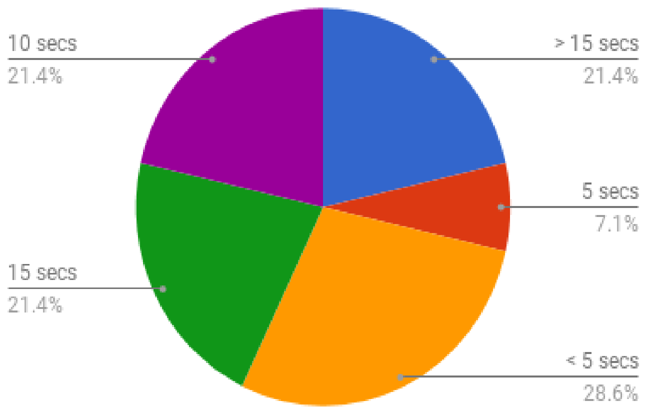


Figure 16: Time taken per task using '/searchrecipes'

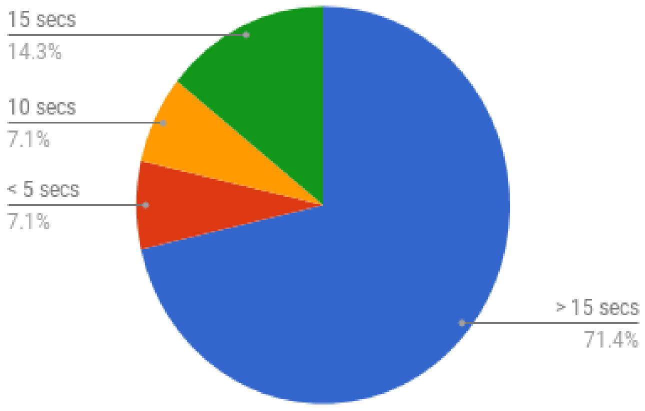


Figure 17: Time taken per task using '/imagesearch'

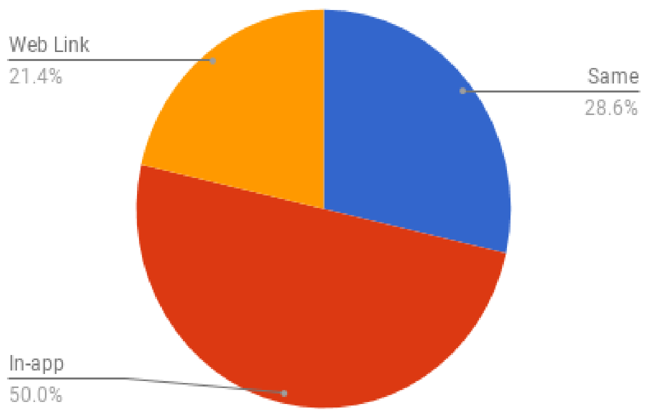


Figure 18: In-app directions vs Web link

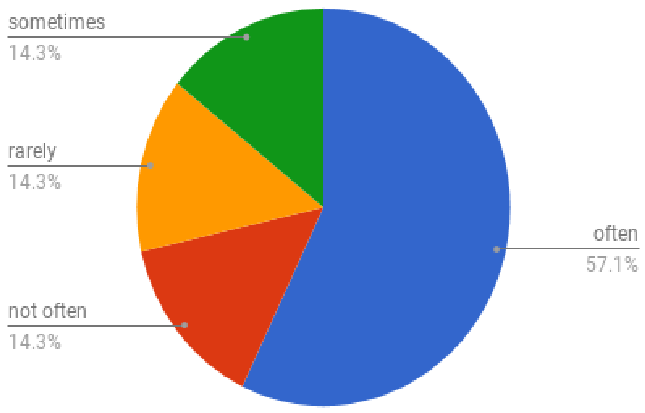


Figure 19: Preference for '/suggest' feature

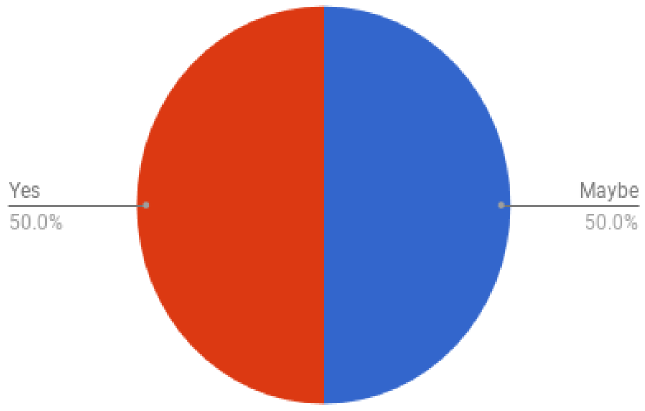


Figure 20: Preference for '/imagesearch' feature

10.6 Compared to version 1.0 (old)

The evaluation process of the earlier version was faulty in itself as no objective and quantitative survey points were present. However, we may roughly compare the user experience of the software usage through the participant's preference of using the software again. In our evaluation process, the feature upgrades of /suggest and /imagesearch were popular from both the quantitative and qualitative survey results and these findings can be extended modestly to conclude that our improved version has satisfied the user's requirement in more ways than one and hence definitely has enhanced the quality with which the initial problem statement was tackled.

10.6.1 *Threats to validity*

It is important to note that some features of our project heavily depend on the underlying platform performance like that of the image recognition software performance, as underlined in the challenges section. We wish to highlight here that other proprietary/open source image recognition software are available which the readers may be interested to experiment this project with.

11. DEVELOPER INSTRUCTIONS

11.1 Development Environment

Follow these steps to set up the development environment:

- Download and install Java SE Development Kit.
- Download and install Intellij IDEA.
- Download and install Gradle build tool.
- Create a MySQL Relational Database System on AWS.
- Create an Elastic Beanstalk application on AWS.

11.2 Developing the Chat Bot

Database

- Load the CSV data file from the Kaggle dataset [3] into the database.
- Create other tables by executing queries given in this file in the repository - src/main/resources/sql_tables.txt

Server

- Clone the repository.
- Run the command 'gradle bootRun' in the terminal to run the application on your local machine.
- Check out the service by running 'curl localhost:8080'.
- Run the command 'gradle build' to generate the WAR file.
- Upload the WAR file in the Elastic Beanstalk application to deploy the chat bot.

Slack App

- Create a Slack app through the Slack API website.
- Install the app in your workspace to test it.
- Add all the necessary slash commands and give the URL to that of your deployed AWS application URL.

11.3 Using Cookbetter 2.0 - Installation

The chat bot has been installed in the following Slack workspace - http://tiny.cc/g9jxsy. To test the chat bot join the workspace either from your mobile device (slack app) or desktop or web (browser) and try out the slash commands. Refer 'what we did section' for the slash commands.

12. REUSE

12.1 GoogleImageFetcher

A simple code that can crawl into google images given a search query to fetch an image. Google generic image API was discontinued hence we had to do this and we feel this is reusable for various purposes.

12.2 RecommenderSystem

Though the recommender system presently works on Recipe and Ingredient object the same can be easily substituted with other domain objects for similar recommenders system such as Movie and Ticket.

12.3 Utility

The methods that have been used more than twice and can be used across projects independent of domains are available in Util.java and at few other places in the code base.

13. GITHUB

https://github.com/snaraya7/cook-better

14. FUTURE SCOPE

We see the following scope for future enhancements in this chat bot:

- **Dietary Planning**: Since we have an exhaustive list of recipes along with their nutritional content (such as protein, calories, fat, etc.), a daily dietary planning feature can be developed. The user can be allowed to set daily targets such as 'I want to consume 50 grams of protein a day' or 'I want to consume less than 2000 calories a day'. A meal plan can be generated for the user suggesting which recipe the user should cook for breakfast, lunch and dinner in order to reach the set goals.
- **Performance Optimization**: There is scope to optimize the performance time of image-based recipe searching. The image recognition API recognizes unnecessary background objects in the image such as refrigerator, tray, etc. By doing an initial level of filtering of such objects in the image, the time it takes to validate if the recognized object is a valid ingredient in the database can be reduced.
- **Retail Integration**: When we recommend a recipe based on user's available ingredients, we track the missing ingredients. Using this data, while recommending a recipe, an additional option can be added to allow the user to purchase these missing ingredients through the chat bot, by integrating with websites like Amazon, Instacart, etc.

15. CONCLUSION

The arduous task of searching for recipes that meet our personal dietary restrictions has been made easy using the initial project idea of the "Cook Better bot" in a very rudimentary fashion using a recipe-search application integrated into the easy-to-use Slack interface. In the current phase of software development, our primary focus has been on improving every aspect of the system based on the feedback of the previous version. Firstly to improve the intuitiveness searching recipes, we have implemented an image and natural language-based input that allows the user interaction with the system to be more flexible, while on the other, the output has been decided to be made more engaging by involving user involvement (feedback rating) which inturn helps users to narrow down on a recipe by taking other users' feedback. We have also dealt with visualizations of user pattern of the application and understood how powerful the visualizations if extended with the a versatile database could be. Further, the recommendation process itself has been proposed to be overhauled with a multi-factor-based searching algorithm.

All the the features have been developed and tested in a strict conformation to established software development principles. Finally, to validate the said improvements we have put forth a comprehensive evaluation that takes the feature specifications made in this design document as the criteria for assessment of our improved software.

References

[1] Amazon. *Amazon Web Services*. https://aws.amazon.com/. [Online; accessed 3/22/2018]. 2018.

[2] Tyner Blain. *Use Case Points Calculator*. http://tynerblain.com/downloads/UseCasePoints.xls. [Online; accessed 3/22/2018]. 2018.

[3] Kaggle. *Epicurious Kaggle Dataset*. https://www.kaggle.com/hugodarwood/epirecipes/data. [Online; accessed 3/22/2018].

[4] *MySQL*. https://www.mysql.com/. [Online; accessed 3/22/2018]. 2018.

[5] Pivotal. *Spring Boot*. https://spring.io/guides/gs/spring-boot/. [Online; accessed 3/22/2018]. 2018.