Boozle Documentation
CS 428
Spring 2014

Michael Willingham (willing3)

Jeff Bailey (bailey27)

Jason Termaat (termaat1)

John Boctor (boctor2)

Akash A Binjrajka (binjraj1)

Rishi Chitkara (chitkar1)

Table of Contents

Description

Have you ever wanted to be a bartender at a party? Did you just turn 21 and have no idea how to make an alcoholic drink? Do you want to discover new alcoholic drinks that you have never tried before given the ingredients you have on hand?

By only knowing the ingredients you have in your apartment, this android application will bring the expertise of a bartender to the palms of your hands! In addition, this android application project could be extended into an automated bartender, which will make drinks that you send to it via the android application! How cool would that be?

This project is to design and create an android application that will serve a few college party goals:

1. The android application will serve as a search engine for alcoholic drink recipes with as clean and simple an interface as possible; you will be able to type in the name of a drink, view related recipes, click on a recipe, and then see the ingredient list.
2. Give the android application a list of ingredients you have at a party, and determine what drinks you can make with your ingredients

Process

The process that we used through the course of this project was shaped by a discussion our experience of using different software development processes through the course of *CS 427 (Software Engineering)* and our internships. This lead us to a combination of waterfall and agile development. We picked the most viable concepts of each of the development techniques and implemented during the development of our project.

The most influential factor of the ones described above is iterative development. We picked this process from the agile development techniques. It was chosen over the traditional waterfall model as it allowed us to set out clear user stories that we will tackle during an iteration and what improvements will be made. At the end of each iteration we got feedback from our grader, evaluated our progress and restarted the process of setting new goals, tackling new user stories and making improvements for the next iteration. This worked extremely well for us as it enabled us to take a step back and evaluate the overall goal of our project at the end of each iteration and make adjustments in case we were going off track. In addition we were able to incorporate the feedback from our grader after the iteration and present it to him by the next iteration. The high number of iterations, meetings and feedbacks enabled us to ensure that we stayed on track in terms of features, usability and timeline for our project.

The next process that played an integral part in our project was pair programming and collaborative development. Instead of rotating pairs for each meeting we set up a constant set of pairs at the beginning of our project. This enabled each member to get to know his partners coding style and strengths. In addition we ensured that the partner who was weaker at a particular concept was the one who writes the code for it while the more experienced programmer checks it in real time. This allowed both the members to expand their skills and work on their weaknesses. Also, the set pairs were not left to drift apart as we maintained the set meeting times of the group. This enabled all of us to still sit and communicate with each other while programming in our pairs. This mimics the real world situation of a agile software development company. It maintains set working hours and an office that enables all employees to communicate with each other while working on their projects in pair programming style. We were able to successfully develop collaboratively through the use of a version control software, GitHub. This allowed each member to check out the code from the repository and work on it on his own machine, then commit back to the repository. This helped each of us look at the changes that were made to the code, solve conflicts and work remotely.

Testing played an important role in our project. We used test driven development for most aspects of our application. This enabled us to figure out the details and requirements of our project by first writing tests that we would possibly need. This also made it increasingly easy to ensure the correctness and efficiency of the code we wrote for the application as the tests for it were already written. More specifically, we used advanced testing as a form of test driven development. This included two forms of tests, parameterized and mock tests. Parameterized tests in our case enabled us to generate a list of all possible ASCII characters and pass them to the ranker. This ensured that random, unexpected characters from the database would not crash the ranker. The second form of testing was mock tests. This entailed creating mock databases to test the functions that we were performing to the data and ensure their correctness and validity. This was essential as our database was too big to perform a large set of tests on it in a timely manner. Running the tests multiple times to check the validity of the methods became efficient through the use of mock testing.

Once the code for each component of the project was written an essential XP principle that came into play was refactoring. Refactoring the code was the most followed XP process of our project as after each iteration we would look over the code that was written and extract, rename, move or modify it in order to make it simpler without altering any of its functionality. This was extremely beneficial as it made our code more organized, readable and understandable. This made it easier to present our code and its functionality to the grader or any other person.

Requirements and Specifications

Our project went through a lot of revisions as we went through the numerous iterations. At the end of it we met most of the requirements we initially started with though there were a number of new features and capabilities which were added along the way. A break up of user stories/ requirements by each iteration is as follows:

Iteration 2*

- Scrape multiple websites for drinks and store the data into a database
- User is able to see information about a drink
- User is able to like and dislike drinks
- Have a loading screen for the app

Iteration 3

- User is able to search for drinks
- Have the capability to search based on ingredients being required or optional
- Have twitter integration and hence user is able to tweet drinks
- Search field should have autocomplete
- Only allow ingredients to be added to a drink if the ingredient is predetermined

Iteration 4

- Use the BM25 search algorithm to search the database
- Have a 'liked' drinks and a 'disliked' drinks list
- User is able to comment on drinks and see the comments of other users as well
- Have a list of the most popular drinks
- User is able to ask for a 'Random Drink'
- Clear distinction between when a user is able to search and when not
- User is able to edit search criteria once search is made

Iteration 5 and Iteration 6

- User is able to look through random drinks
- Easy app navigation
- Search for alcohol types regardless of brand of alcohol
- Clean up the data used to make it more uniform

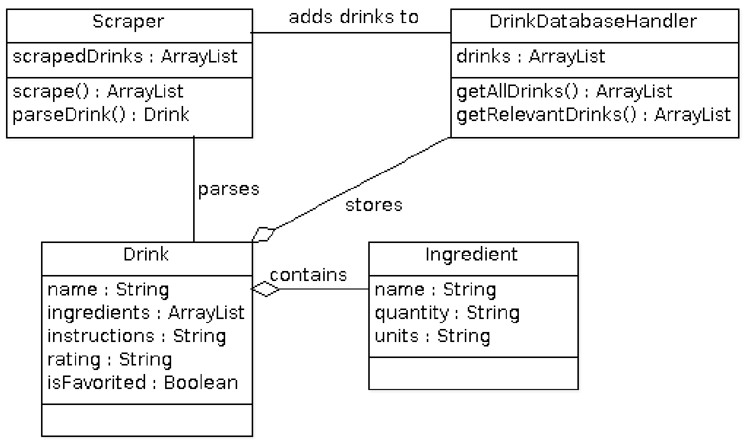
**Iteration 1 is not mentioned since no work was done on the application done and it was just a planning phase*

Architecture and Design

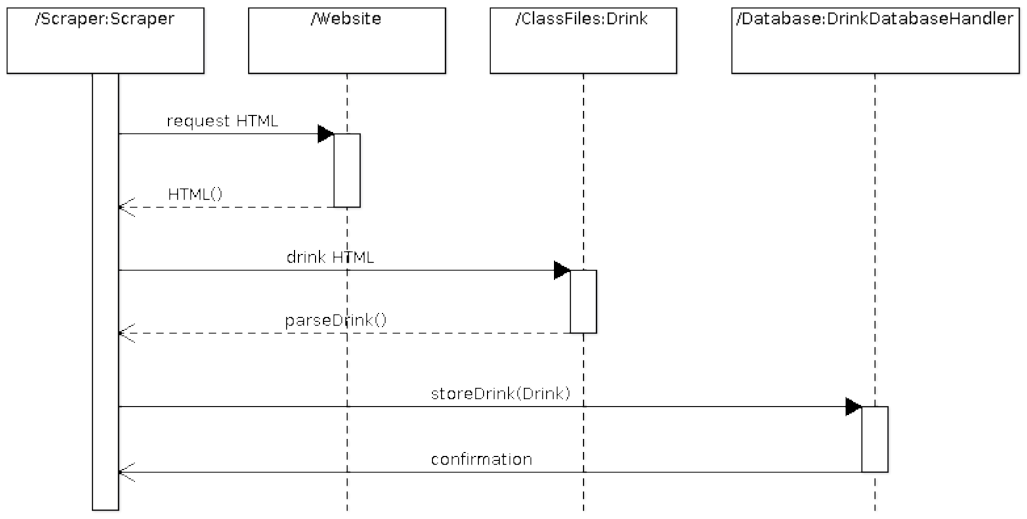
Scrapers

The first part of our project was to collect drinks recipes from well-known websites. This allowed us to populate our application with a large amount of data without having to hand-write recipes ourselves.

In order to do this, we used a framework called JSoup to create DrinkMixerScraper.java, DrinkOfTheWeekScraper.java, and GoodCocktailsScraper.java to scrape drinkmixer.com, drinkoftheweek.com, and goodcocktails.com. Each of these scrapers take the HTML of the drink recipe pages on the website, and turn them into a usable data structure (Drink.java) to be stored in our database. Once we run all the scrapers, we collect all the drinks and store them into our database using DrinkDatabaseHandler.java. You can see more about the DrinkDatabaseHandler.java in the Database section.



UML Class Diagrams for Scrapers



UML Sequence Diagram for scraping a website and storing drinks

Android Front-End

A few different frameworks were used in the development of the front-end of the app. In developing the front-end itself, native android XML was used. This was chosen over webviews because, though webviews are more pluggable, XML is internally optimized and consequently would run quicker on the device itself. iOS development isn't in our plans for this, so the pluggability of webviews was really not an applicable potential benefit.

A second framework that was used was a side navigation bar library. There is a way to build this from scratch using native android fragments and views, however one of the team members was familiar with this library, and suggested it as a simple plug to have the system for navigation in place instead of taking quite some time building it.

A third framework that was used was a twitter wrapper to handle the twitter interaction for the drinks. The twitter API is notoriously frustrating to work with, so similar to the decision to use the side navigation library, we decided to use a fairly widely used Java wrapper library to handle interacting with that instead of writing our own code from scratch.

A fourth and final framework we used was Socialize, a comment thread/server-based API that's pluggable into android apps. This adds a lot of aspects to the application - first and most notable (to the user) is the comment thread that exists for each individual drink. This also allows likes of drinks, and displays the number of views, number of likes, and number of comments per drink. More interestingly, the system is extensible to view the aggregate set of views, likes, and comments, with meta-data available as well. Due to this, it opened up an aspect of the app to view the 'most popular' drinks, sorted based on views, likes, comments, and times thumbs upped.

The first thing that happens when the app is loaded is the loading activity is launched. That then takes the base context and sets the content to the loading layout. It then gets references to the base context and actionbar, the context for future interface work and the actionbar to hide the title (purely for the sake of aesthetics). Android doesn't have native .gif handling, so it then creates an AnimationDrawable to display the loading screen .gif of a filling up beer. Finally and most importantly, it spawns a thread that calls the DrinkDataBaseHandler to get all drinks, which fires over to the home activity when it's done.

Home starts the same way - setting the layout and getting the same references. It then gets the list of drinks from the loading activity that just finished running. The linearlayout that's in the baseview has an ontouchlistener set to handle swiping to create the menu, which is a simple extension of the base ontouchlistener. The back button is overridden here purely to avoid creating a new intent to go to loading. The native screen upon getting to this activity is the linearlayout cleared, adding only some instructions, a few navigation buttons, and an imageview.

Search is one of the main aspects. It creates a dialog through which the user selects what type of search they want to do. For by name, it gets the name of a drink and sends it to the search engine, for by ingredient it gets a list of optional and required ingredients and sends those to the search engine. When looking at search results, an onclicklistener is configured that will make a popup for info about that drink. It first finds the appropriate drink object, then displays the data linked to that drink in the dialog. There's a comment thread view at the bottom which interacts with the appropriate entity on a server, asynchronously. The twitter button will initiate TwitterInteraction, which, if a user hasn't signed in, will initiate an OAuth process to do so, but otherwise will send a tweet to their account about that drink. If the user interacts with the thumbs, it immediately syncs it with the database and the local list of drinks, and if it were a thumbs up, it fires a thread to sync that fact with the server.

There are a few menu options, all of which are available in the side menu when there are displayed results and some of which are on the no results view. The side menu itself is toggled to be visible on a registered swipe from the ontouchlistener linked to the parent view or if the acitonbar icon is clicked. Go home on the side menu fires an intent that sends it back to the no results view, getting rid of the list of results. I'm feeling hammered displays the popup of a random drink. Liked drinks and disliked drinks iterate on the list of drinks and display those that are thumbs upped or down, respectively. Most popular spawns a thread that queries for the most popular drinks on the server, sorting them by various means, and displaying them on a dialog.

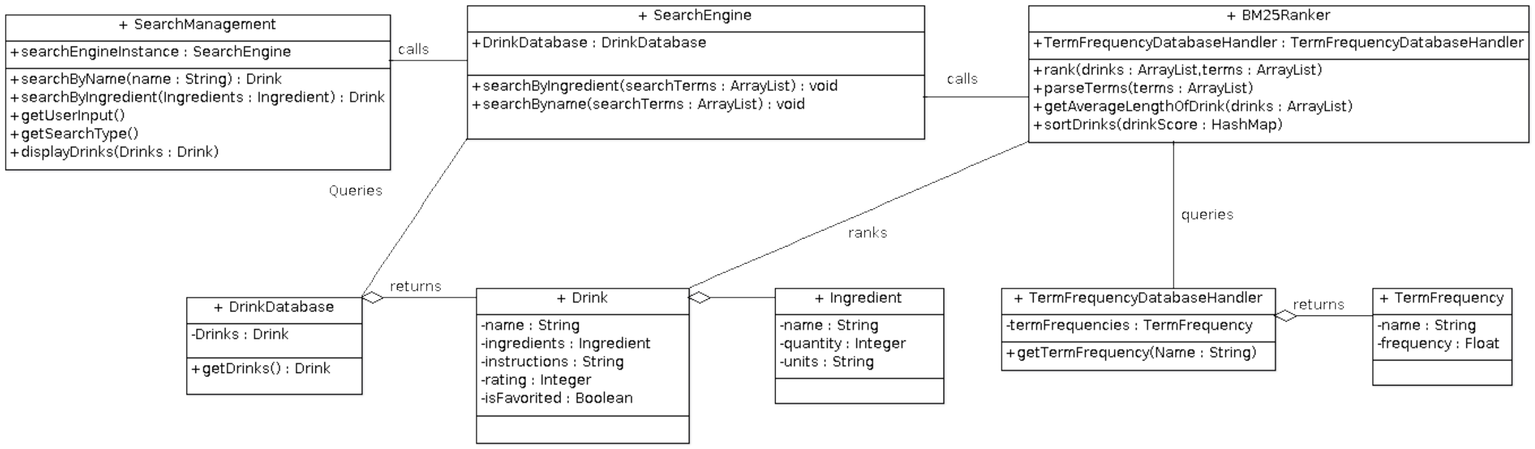
Search Engine & Ranker

The search engine component takes the user's search query, takes what type of search they want, queries for relevant drinks, and then ranks the drinks to put the most relevant drinks on top. Therefore, if a user were to search for “Gin and Tonic,” drinks related to gin and tonic would show up towards the top.

The SearchManagement class interacts with the user. It takes the user's query, and what type of search they want to do. The SearchManagement class and passes this information to the SearchEngine class.

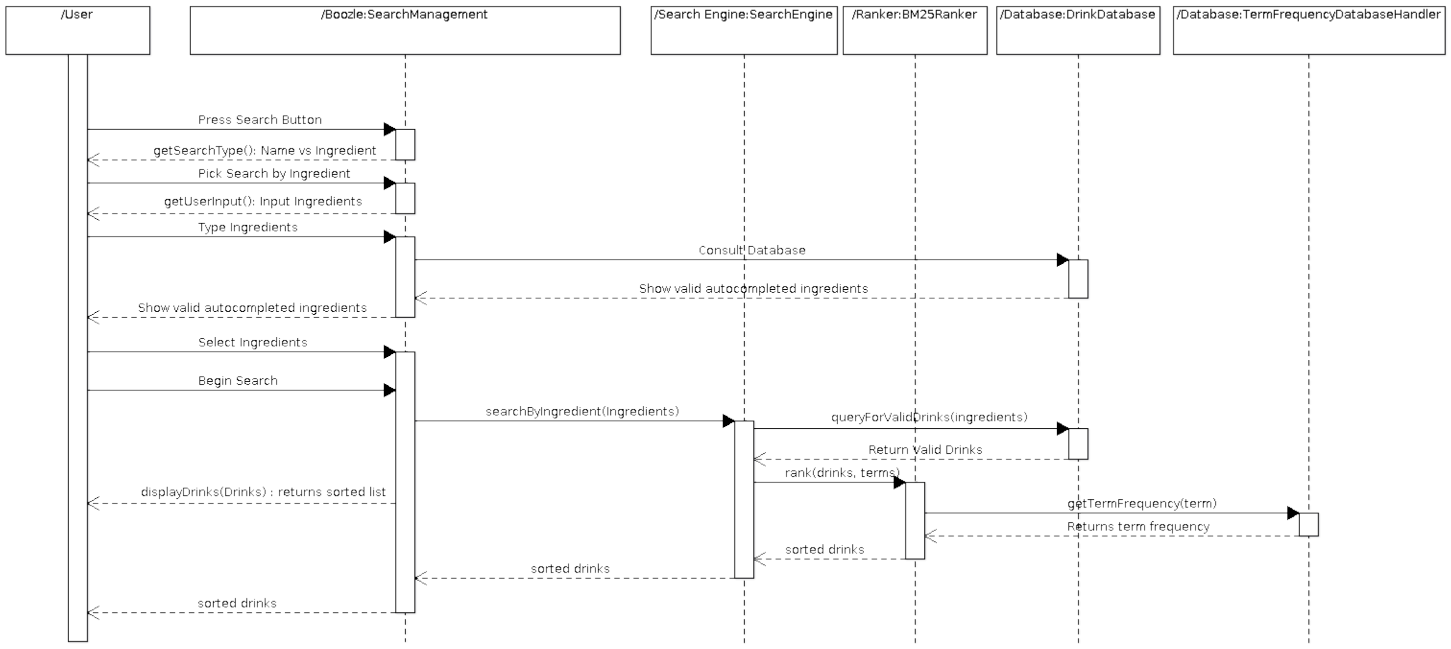
The SearchEngine class takes the user's query and search type, and queries the drink database for relevant drinks. Specifically, it queries for drinks with names that are related to the query. It takes these releveant drinks, the user's query, and the search type, and passes it to the BM25Ranker class.

The BM25Ranker class takes the search terms, relevant drinks, and search type, and uses the BM25 ranking algorithm to rank drinks. Part of this algorithm uses document frequency, term frequency, inverted document frequency, average document length, and more. To make this process efficient, we utilize the a database to store term frequencies. To interact with this database, we use the


+ SearchManagement
+searchEngineInstance : SearchEngine
+searchByName(name : String) : Drink
+searchByIngredient(Ingredients : Ingredient) : Drink
+getUserInput()
+getSearchType()
+displayDrinks(Drinks : Drink)
calls
+ SearchEngine
+DrinkDatabase : DrinkDatabase
+searchByIngredient(searchTerms : ArrayList) : void
+searchByname(searchTerms : ArrayList) : void
calls
+ BM25Ranker
+TermFrequencyDatabaseHandler : TermFrequencyDatabaseHandler
+rank(drinks : ArrayList,terms : ArrayList)
+parseTerms(terms : ArrayList)
+getAverageLengthOfDrink(drinks : ArrayList)
+sortDrinks(drinkScore : HashMap)
Queries
ranks
queries
+ DrinkDatabase
-Drinks : Drink
+getDrinks() : Drink
returns
+ Drink
-name : String
-ingredients : Ingredient
-instructions : String
-rating : Integer
-isFavorited : Boolean
+ Ingredient
-name : String
-quantity : Integer
-units : String
+ TermFrequencyDatabaseHandler
-termFrequencies : TermFrequency
+getTermFrequency(Name : String)
returns
+ TermFrequency
-name : String
-frequency : Float

TermFrequencyDatabaseHandler class. You can see more about this class in the Database section.

UML Class Diagram detailing the classes related to the search engine and ranker

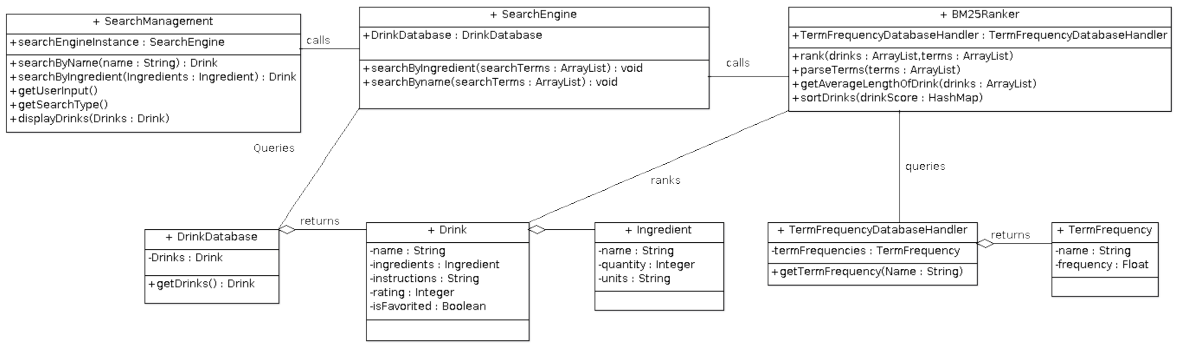

/User
/Boozle:SearchManagement
/Search Engine:SearchEngine
/Ranker:BM25Ranker
/Database:DrinkDatabase
/Database:TermFrequencyDatabaseHandler
Press Search Button
getSearchType(): Name vs Ingredient
Pick Search by Ingredient
getUserInput(): Input Ingredients
Type Ingredients
Consult Database
Show valid autocompleted ingredients
Show valid autocompleted ingredients
Select Ingredients
Begin Search
searchByIngredient(Ingredients)
queryForValidDrinks(ingredients)
Return Valid Drinks
rank(drinks, terms)
displayDrinks(Drinks) : returns sorted list
getTermFrequency(term)
Returns term frequency
sorted drinks
sorted drinks
sorted drinks

UML Sequence diagram detailing a sample use case of “Search by Ingredient”

Database

There were two main databases used in our project. One database was used to store drinks and their ingredients. The other database was used to store frequencies for all the terms parsed from the drinks and ingredients. We interact with the databases by using the DrinkDatabaseHandler and TermFrequencyDatabaseHandler classes. These two classes handled CRUD (create, read, update, and delete) and table creation functions.

The specific database type we used was SQLite. We chose this type because it is a standard in android applications that do not have a large amount of data. Android has a helper class that can be extended called SQLiteOpenHelper. This class forced us to extend functions for initializing the database, populating tables, and updating the tables when a new database version is available.


+ SearchManagement
+searchEngineInstance : SearchEngine
+searchByName(name : String) : Drink
+searchByIngredient(Ingredients : Ingredient) : Drink
+getUserInput()
+getSearchType()
+displayDrinks(Drinks : Drink)
calls
+ SearchEngine
+DrinkDatabase : DrinkDatabase
+searchByIngredient(searchTerms : ArrayList) : void
+searchByname(searchTerms : ArrayList) : void
calls
+ BM25Ranker
+TermFrequencyDatabaseHandler : TermFrequencyDatabaseHandler
+rank(drinks : ArrayList,terms : ArrayList)
+parseTerms(terms : ArrayList)
+getAverageLengthOfDrink(drinks : ArrayList)
+sortDrinks(drinkScore : HashMap)
Queries
ranks
queries
returns
+ DrinkDatabase
-Drinks : Drink
+getDrinks() : Drink
+ Drink
-name : String
-ingredients : Ingredient
-instructions : String
-rating : Integer
-isFavorited : Boolean
+ Ingredient
-name : String
-quantity : Integer
-units : String
+ TermFrequencyDatabaseHandler
-termFrequencies : TermFrequency
+getTermFrequency(Name : String)
returns
+ TermFrequency
-name : String
-frequency : Float

UML Class Diagram Demonstrating the Two Databases

Tests

The tests for our android application are in a separate project called TheInebriatorTest. We separated the android application tests from the main project because it is industry standard for android development. It is industry standard because separating the tests from the main project reduces the footprint of the application on an android device.

Nearly all our tests are android unit tests. They are automated tests, similar to JUnit tests. The difference is that they require an android emulator or device attached to the computer to run.

There are automated tests for the some data structure files, including: Drink, Ingredient, Matching, etc. There are also automated tests for the search engine, including: BM25Ranker and the SearchEngine class. Lastly, there are tests for the databases. These tests cover both the DrinkDatabaseHandler and TermFrequencyDatabaseHandler classes.

All of our tests can be run via a test suite, AllTests. This will run all our android unit tests and the advanced unit tests.

Future Plans

In the future, we have a few more features we would like to add to our application. Many of these features revolve around multiple users and networking. The first feature we would like is to have users contribute alcohol they have at a party to a collection. Once this collection is established, you could see what drinks you could make from everybody's alcohol. Another feature that we want to add is allowing users to view the specific drinks they added to the database. Right now, the only way to see their custom drinks is to view the entire database.

Also in the future, many of our group members would like to use the Boozle application as an interface for an automated bartender. We would like to use a Raspberry Pi or Arduino to build a mechanical system to pour drinks based on the ingredients in our android application.

We have also gotten the chance to explore add ons for our app. An example of this is a widget that we have worked on and included in the final package. It functions as an add on for the application for now with the function of allowing the user to see a random drink from the list of popular ones. It also has a 'try it!' button which acts like a quick drink counter for the user to keep track of how many drinks he has had without having to open an application.

In addition to new developments, our group would also like to release our application into the android app store. In order to do this, we could go one of two ways depending on the user response for this version of the application. The first way would be to set up a server and allow users to create a log in ID and therefore access their liked and custom drinks from anywhere. This would also allow them to share their drinks with friends without the use of the social network features that we have added. The second way would be to clean our database by manually fixing the minor irregular errors caused by lack of uniform formatting of the databases of the websites that we parsed for cocktail recipes.

Personal Reflections

Mike:
Throughout this project, I learned a great deal about agile development and other software development industry standards. Specifically, I learned how important test-driven development can be to ensuring your code works and making refactoring easier. Also, I had my first run-in with UML diagrams. These diagrams allowed me to communicate ideas in a much clearer way, and I will hopefully be using these diagrams later in my career.

John:

This is one of the biggest projects I have done in my college career. And as such I have grown to have a special connection with it. Granted it was not a good one, but spending as much time as I did with it, that was bound to happen. First off, for a three credit hour class I believe that too much was expected from us. Our TA was consistently telling us that we needed to put in 20 hours or so a week to have a good finished product which I found to be absolutely ludicrous. I personally love programming, but having to fix miniscule issues that our TA kept nitpicking at was one of the worst experiences of my life. In regards to the team, I feel that for the most part we had a good team. We worked hard together and got what we needed done complete on time. There were issues with people not coming to meetings very often, but in the end we managed to take care of things regardless. I did learn that I never want to do android development ever again. I'll stick to IOS and Windows Phone.

Jeff:
The project was a lot of fun to work on. In hindsight, there are a few things I would've done differently. Towards the end of our process, we had a lot of issues with malformed scraping of information, which was fairly avoidable. All we had to do was atomically work on the scrapers themselves, testing them thoroughly as they were built, and we would've been able to catch most of these earlier. It would have been easier to fix things individually then instead of needing to re-run the whole scraper process. Additionally, a more defined database system being put in place would've helped save quite a bit of time. Process-wise, we only had a few issues. As described above, more rigorous testing would've been in our best interest. As far as people went, we needed some kind of system to help with attendance of meetings - we had a hard time getting all six people to show up and work together, and that hampered productivity, greatly at times. Other than that, though, it was a good experience. I learned a lot working with the various frameworks we used, gaining some experience with OAuth via Twitter's API, I learned a few new things about android UI work, building a lot of cool parts to the app with them, and I learned a lot about database interactions with SQLite and android together.

Rishi:
I enjoyed working on this project and learnt a lot as it was one of the longer projects that I have worked on. Deciding on the development process and sticking to it was a new concept to me. It helped as the team knew what needed to be done at each stage. This project and the class itself helped blur the lines between theory and application. The UML diagrams were useful in the development of our application and understanding the relation between the elements in our database.

Jason:
This project was a real learning experience. Because this project was on a much bigger scale than most of the projects I work on for school, I ran into a lot of new problems and challenges that I rarely get to see in class. For instance, design decisions made on the second week regarding how data was parsed into a database had repercussions for how that data had to be handled for the rest of the project. This has taught me that it is important that when designing something that is important to a large application, I must make design choices carefully and preferably make them modular and possibly even reversible. I found that even simple functions like unit tests can be obsoleted by new design decisions. For future projects I will try to avoid problems such as this that I encountered by having a clear design for what my code should do from start to finish before I begin writing tests or code.

Akash:
As a CE major this was a class I was looking forward to the most since I took it as a tech elective. This class gave me the opportunity to work on an application from beginning to end and follow an industry level software development process. I learnt a lot from this project. I had never worked this extensively in android before and hence I was introduced to a lot of new things. My team was great- hardworking and very helpful, hence I never felt lost even when we were working on things completely new to me. I got the chance to interact with massive data and the learning curve for database handling was very steep. I had never worked with the jSoup library and that was a learning experience as well. In hindsight, I wish I worked a little more on the UI end of the application since I could have gained some knowledge in that part of the development process as well. Overall, I had an amazing time working with my team and I am proud of what my team has achieved.