≡SpeedyChef

Milestone 7

Project Owner: Alia Robinson
Scrum Master: Chris Budo
Developers: Steve Trotta, Larry Gates, Chris Budo, Alia Robinson

Table of Contents

- Introduction
- Change Log
- Epic Level User Stories
- Feature Level User Stories
- Low Fidelity Prototypes
- Evaluation Plan
- Results
- Revised Low Fidelity Prototype
- High Fidelity Prototypes
- Coding Progress
- Continuous Integration
- Feature Completed
- Value Proposition Canvas
- Business Model Canvas
- Usability Study
- Usability Study Improvements

Introduction

Preparing a meal for a family, group, or event can be a time-consuming process. Whether one is preparing for a party or simply making dinner for a family, no one wants to waste unnecessary time on cooking. Inexperienced cooks may also struggle to finish all of their dishes at the same time, so that they may be served hot and fresh.

Our proposed solution is a software product to remove some of the hassle from this process. With regard to user convenience, we have decided that this idea is best suited to a cross-platform mobile app. The app will have the option to choose a set of recipes from several different online sources. Given this list of recipes, the application will suggest a time-optimized plan for preparing a meal. The user will be instructed to make the best possible use of their time by performing concurrent actions such as chopping vegetables for one dish while another is cooking on the stove. The end result should be that all of the selected dishes are ready to serve at the same time.

Several cooking-related web services and apps exist, performing tasks such as finding recipes based on what ingredients the user has, or evaluating food based on its nutritional value. However, we have been unable to find a product that performs the service described here. While the main focus of this application would be to help a user minimize their cooking time, other features could be added such as displaying nutritional information and suggesting supplemental dishes. The only effort required from the user is selecting a set of recipes- all of the other planning is done for them. Thus, this technology-based solution relieves the stress and pressure of cooking and allows users to easily explore new recipe options.

Change Log

Milestone 3 - 4:

While developing the *Plan* screen, fitting all the buttons in a single row was going to be difficult. As suggested by Chris to Larry, we added a horizontal scroll bar as a way to find a day of the week easily. Also, this allowed the buttons to be larger on the screen. To maximize the size of each day button, a separate shifter was added to the design, to advance the calendar one week at a time. By adding this extra location, the designing of the button and layout was simpler and more user friendly. This also allowed space for displaying the month and year while passing through dates.

The next change was to the *Meal Designer* (Meal Planning (2) prototype), where an additional button was added. The button added, Save Meal, will allow the user to return to the calendar page after setting up a meal plan.

One other change involved the recipe walkthrough feature, which displays progress bars at the top of the screen for the timers currently counting down. Due to the constraints of the system, we are limiting the maximum number of active timers to 5.

In the next milestone, the *Meal Planner* daily page will be adjusted to display the assigned meals for the day. Additionally, the *Diner* counter will be changed to a horizontal layout instead of vertical layout, to help increase space. As the UI components for this milestone were completed, the need for this change became apparent. We also plan to reevaluate the overall color scheme of the UI and decide on the look and feel. The Preferences page will also be updated to better fit with the current style.

Milestone 4 - 5:

For the *Meal Designer* page, the number incrementer for the number of diners was changed to a slider. The slider was easier to handle, since the number incrementer object could not be rotated, wasting a lot of space.

For both the *Meal Designer* and *Calendar* page, the buttons that should be generated on the inside of the scroll view are not dynamically loaded yet, due to the fact of the connection between the back end and the front end. The stored procedures are all ready to go but the server was having trouble up until Thursday (10-29-15), but EIT gave access to a blocked port. Now setting up the server on campus is in progress. In the Feature Completed section, the parts of the

stored procedures and API will be displayed, and an output. The stored procedures were tested and checked to ensure they work.

For the default *Search* screen, the choice dropdown has been replaced by a *ContextMenu* object for ease of access and size of text (basic usability). In addition, a secondary option will soon be added to give the user a choice between ascending and descending view order based on the category.

EIT has been contacted and we are now able to access the port that was previously blocked, we are now using the azure database, and Chris is currently working on pushing the api into the cloud (and setting up continuous deployment).

Milestone 5 - 6:

The *Calendar Page* and *Meal Designer Page* both required **synchronous** connections with the API calls instead on asynchronous connections. This was very important to keep one page collecting data before data changes were made to the database. These changes would happen but to load the buttons with recipe or meal information after a meal/recipe was removed or added, the page or API call would have to be activated again. This was hard to realize since these implementing these API calls were a new idea in Android. While asynchronous is slightly faster, synchronous will provide correct results more often than asynchronous calls.
The **Bookmark Recipe** button was replaced with **Remove Meal** as a way to remove the amount of buttons on the page, and remove a button that was not going to be implemented anytime soon.

For the *Calendar Screen Page*, adding a **START WALKTHROUGH** button. This seemed the best location to access a meal's walkthrough, and requiring no additional button counts once you find the meal on the day. Also the layout of information of the meal was changed, such as using an image of a person to denote the count.

For the *Meal Design Page*, when recipes are added to the meal, a remove button is added so that it is easy to remove the connection of a recipe to a meal. By clicking the remove button, the recipe is immediately unrelated to the meal, despite the user clicking the back button to cancel the operation.

The *Search* screen hasn't changed much, but a small change is planned to include the choice of ascending and descending order. Also on the horizon, if time permits, is a more uniform and aesthetic design for search results listed general color and content will remain, but size and proportionate layout of the content will change slightly in ways still undetermined.

The *Menu* is being changed as we speak to include slide-drawer functionality that was suggested to us by our subjects/participants in the usability study conducted on 11/5/2015. Many android applications have this technology, and while somewhat involved to implement some teammates believed it to be an important enough feature to include in our MVP.

Continuous Deployment has been set up, as well as a code review.

The *Preferences* page now is composed of three separate pages which can be switched between using three buttons along the top of the page. These pages include all of the planned functionality, although the approach to some of these tasks has changed from the initial design.

Milestone 6 - 7:

All changes are addressed in the *Usability Study* Report below.

Epic Level User Stories

Reduced Meal Preparation Time

Preparing meals with multiple dishes takes too much time. Students generally responded that they would be more willing to spend time preparing complicated dinners if this took less of their time to accomplish.

Improved Coordination of Multiple Dishes

Cooks find it difficult to finish preparing multiple courses at the same time. Most students indicated they struggle with this, and some expressed that this is especially difficult when there is a large difference in cooking time.

Provide A Single Curated Recipe Source

Most students choose to find new recipes using Google, rather than consulting one specific website. While this is not a problem, it presents an opportunity to offer a one-stop solution for recipe discovery.

Easy Access To Nutritional Information

Many cooks do not take the extra time to research the nutritional value of the food they prepare and consume.

Allow Saving of User Preferences

Users should be allowed to save data relevant to the equipment they own, the ingredients they cannot consume, their level of cooking experience, and other relevant data.

Feature Level User Stories

Core Features

Searchable Recipes

As a young adult looking to try something new for dinner tonight, I want to be able to search for recipes directly through SpeedyChef, allowing me to access lots of diverse recipes in one place and apply the other features of the app.

Conditions of Satisfaction:

Have an easily identifiable "search" screen on the app.
Effectively search the database for recipe tags or keywords that the user inputs.
Suggest tags or keywords as the user is typing.

Equipment Capabilities Saved (Preferences)

As a college student with a set line of kitchen utensils and appliances, I want to have the ability to log what equipment I own / have available so that I can apply that list to my search of recipes, filtering out the ones I can't physically make because of restrictions in equipment.

Conditions of Satisfaction:

Have intuitive screen to enter current kitchen utensils and appliances owned.
Search recipes or limit recipes by available means to cook.
Expand search past these limits, the system telling you what you must procure to make the dish.

Experience Logged (Preferences)

As a middling-level chef with average or below average experience, I want to know what meals I can and cannot cook depending on what my experience will allow me to comfortably do. This means logging initial experience level or taking a composite of my experiences and using this rating to find recipes I would be capable of (within a reasonable time at least).

Conditions of Satisfaction:

Have quality indicators of "experience" that can be somewhat easily decided upon and averaged
Evolve these over time with the submission of data (perhaps - think of taking the time it took to cook and averaging it with the rest of the data using some heuristic of "experience")
Search recipes or limit recipes by experience level
Recommend helpful tips or tricks to improve

Picky Ingredients Searching (Preferences)

As a lactose intolerant human being, I need to avoid ingredients such as milk and cheese. SpeedyChef can help me with this. I can submit all the ingredients that I can't consume into a

blacklist - that list will filter out all the recipes that contain illegal ingredients when I search for new meals to make.
Conditions of Satisfaction:
Simple, quick UI for adding and removing illegal ingredients
Search parameters that use the blacklist to filter

Cross - App Timer

As a busy, full-time worker, I need to constantly expedite my tasks throughout the day and multitask whenever possible. A timer display that can be accessed in some way, even while in other applications would help me with this goal - the timer would constantly apprise me of when my instruction step or meal cooking in general will be finished.
Conditions of Satisfaction:
Easy to read and understand interface for time that isn't inconvenient to look at / deal with
Ability for timer interface to always be seen or accessed while on the device
Helpful chime or reminder when crucial step-times are arriving.

Advanced Meal Planner

As a busy, working parent, I want to be able to schedule meals for my family ahead of time and decide on exactly what I'll need for my meals (that way I don't waste a bunch of food too). I can do this with Speedy Chef, which allows me to select recipes for the rest of the week or even further. This allows me to easily plan in advance what I will be cooking as well as what ingredients I will need.
Conditions of Satisfaction:
Easy to navigate / ergonomic display for selecting this week's recipes
Perhaps the ability for multiple people to enter to recipe list / shared account or planner
Auto erasing / garbage data collection for old days / weeks (but save relevant data on past days)

Recipe Walkthrough

As a parent of multiple children, I want to be able to efficiently prepare multiple dishes for our family's dinner. I would benefit from instruction on how to complete the recipes, and it is helpful to me if they are all finished at the same time.
Conditions of Satisfaction:
Displays easily navigated and understood list of steps for completing all recipes in a meal
Offers time estimates for completing each step
Aims to complete the meal in as little time as possible
Ensures that dishes are completed at the same time

Additional Features

Grocery List

As a working parent, I want to be able to make a single grocery trip on Sundays to buy enough food for a whole week of dinners. SpeedyChef saves me time by creating a grocery list for me based on my meal plan for the week.

Conditions of Satisfaction:

Auto-compiling shopping list (selection and unioning of similar / identical ingredients)
Convenient and intuitive display for the shopping list itself
Ability to make executive decisions regarding what is or is not on the list.
Ability to "check off" items on the list once the user has found them.
Integration with store finder feature for user convenience.

Timing Decisions and Overrides

As a growing student of the chefly arts, I will be developing faster and faster meal preparation skills that will affect my estimated time of completion. On top of this, I know my kitchen well and should be able to override some of the instruction step times to times closer to what I would be achieving. SpeedyChef allows me to go into a step time and reselect a time more appropriate that will factor into the estimated total cooking time.

Conditions of Satisfaction:

Affordable / visible step-time adjuster
Triggers or procedures that log these new step-times for future use / calculations

Store Finder

As a new member of the Terre Haute community, I don't necessarily know what local stores I can get groceries from for what I'm making. SpeedyChef contains a shop-finding feature that I can use to locate local stores that sell the ingredients I need for my recipe. I just choose a recipe and at the bottom click the "Where to Buy" button and a list of relevant stores will be displayed.

Conditions of Satisfaction:

Accurately receive the correct store information based on GPS coordinates and if possible inventory selection
Efficient and logical ordering of store information and locations
Non-intrusive location for the "Where to Buy" button and easy path to return to recipe screen

Nutrition Guide

As a healthy individual that likes to stay fit and well-nourished, I need a method of tracking my daily intake of essential foods. SpeedyChef takes this into account once I select this preference

for my account. It tracks, based on the recipes that I choose, what kinds of nutrients I have been getting from my meals. It displays this information simply, but also uses in its choices of sensible future meals.

Conditions of Satisfaction:

Procedures for determining nutritional content of each recipe / serving and the capabilities of saving this information over time

Simple to read UI for displaying nutritional information and erasing / restarting the collection of this information

Easy way to either search using nutrition or not with that faction depending on the client (most people seem not to search for meals based upon health)

Adequately weighted factor for nutrition in the search algorithm, but not too weighted as to pick unhelpful meals for other categories / factors

Dinner Bell

As a parent of a busy family, I find it difficult to have everyone show up to dinner on time. It would be wonderful to be able to notify all members of my family that dinner is ready and they should come to the table.

Conditions of Satisfaction:

Push notification to devices owned by family members

Easy to send notification when dinner is done

Set up list of family members to send notifications to

Recipe Suggestions

As a young adult stuck in her daily routine, I want to be able to discover new recipes without any extra effort on my part. SpeedyChef helps me do this by suggesting recipes for be based on my preferences and needs, allowing me to expand my list of recipes that I am comfortable with.

Conditions of Satisfaction:

Display a list of suggested recipes when prompted by the user

Optionally suggest a recipe from the meal planning screen.

Favor recipes with ingredients that the user seems to like.

Do not suggest recipes with ingredients that the user cannot eat.

Favor recipes that are tailored to the user's nutritional needs. For example, if the user habitually does not consume enough calcium, suggest a calcium-rich meal.

Substitution Suggestions

As a young person on my own for the first time, I often do not have enough time or money to purchase food items that I will only use for one recipe. I want to be able to easily find acceptable

substitutions for ingredients that I don't have access to. This will make it easier for me to cook with limited time and resources.

Conditions of Satisfaction:

Display a list of suggested substitutions when prompted by the user

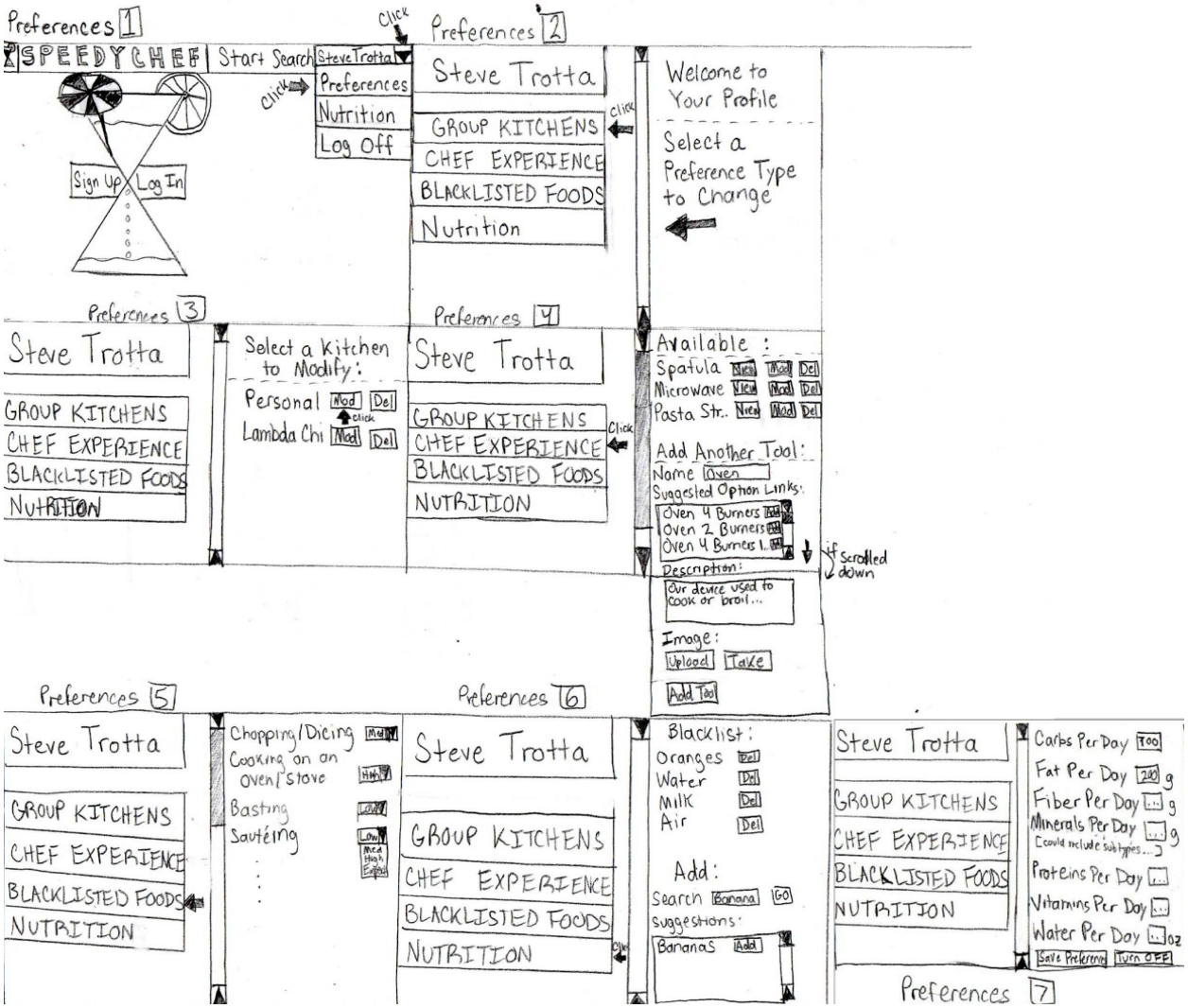
Allow the user to access this list when looking at a list of ingredients

Calculate an appropriate amount of the new ingredients to suit the recipe.

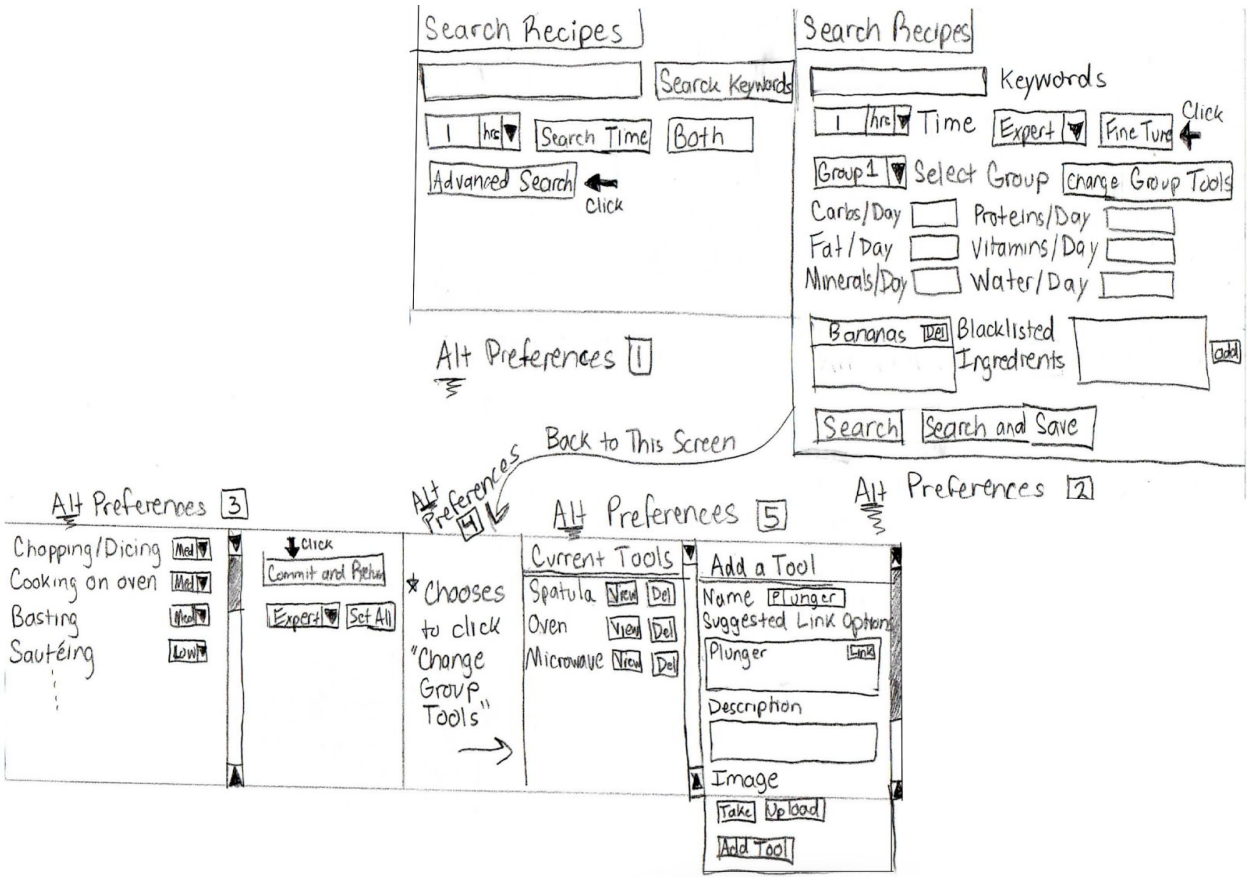
Prototypes

Preferences

Prototype One



Prototype Two



Prototype Three

Allergies

- Peanuts
- Tree Nuts
- Milk
- Soy
- Gluten
- Eggs

Skill Level

Novice — Expert

Set Preferences

- Cooking Facilities
- Kitchen Tools
- Favorite Cuisines
- Allergies
- Skills

Prototype Four

Kitchen Tools

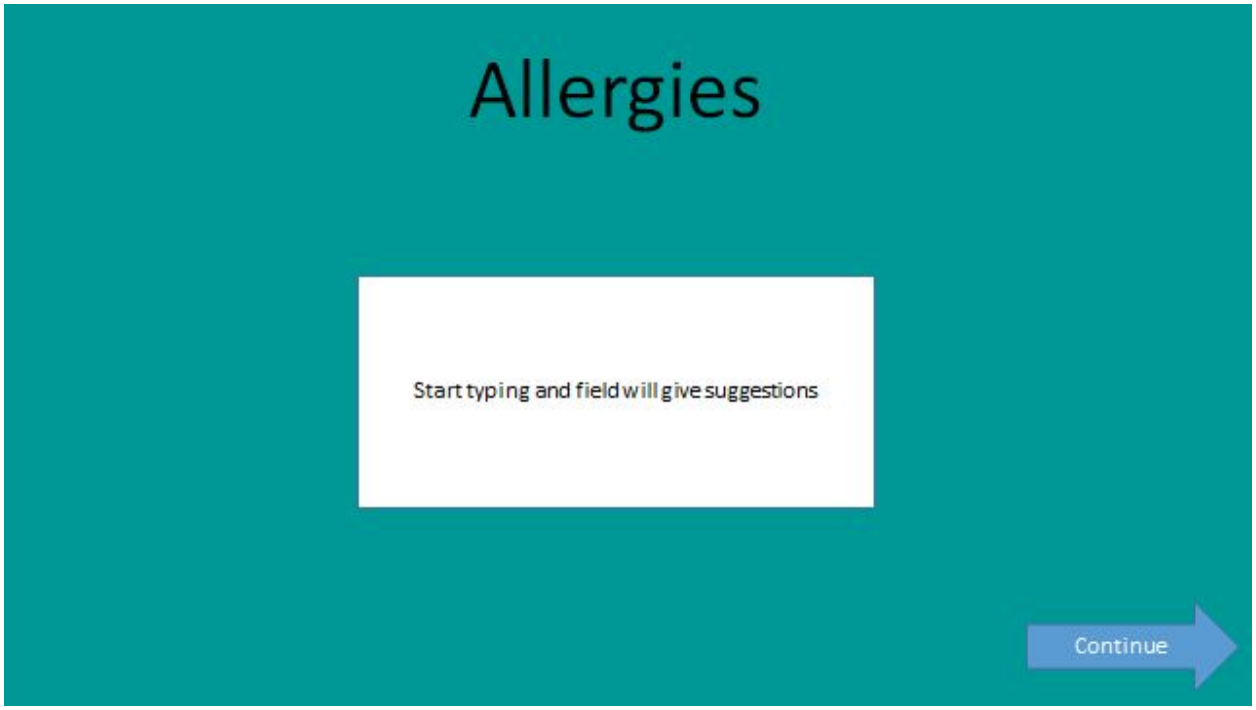
- [ ] Chef Knife
- [ ] Crock Pot
- [x] Sauté pan
- [ ] Sauce pan

Continue

Favorite Cuisines

- [ ] Chinese
- [ ] Italian
- [x] American
- [ ] Sushi
- [ ] Fusion
- [ ] Mexican

Continue


Allergies
Start typing and field will give suggestions
Continue

Skill

Continue

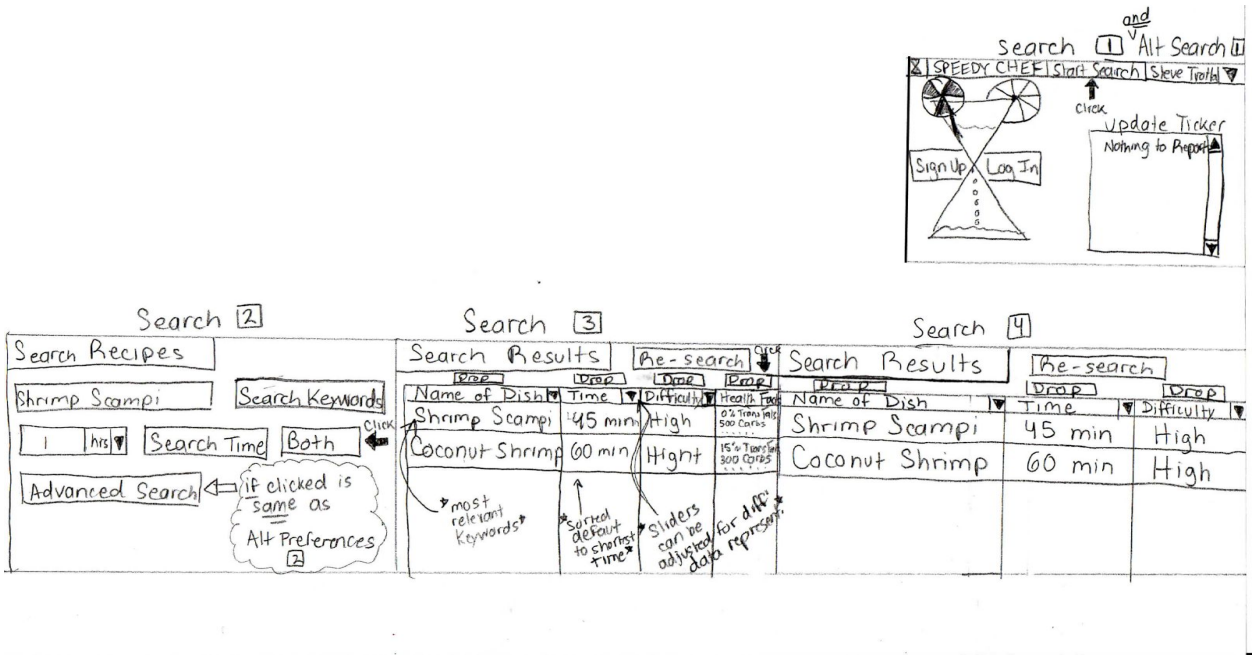
Preferences

- Cooking Facilities ✓
- Kitchen Tools ✓
- Favorite Cuisine ✓
- Allergies ✓
- Skill ✓

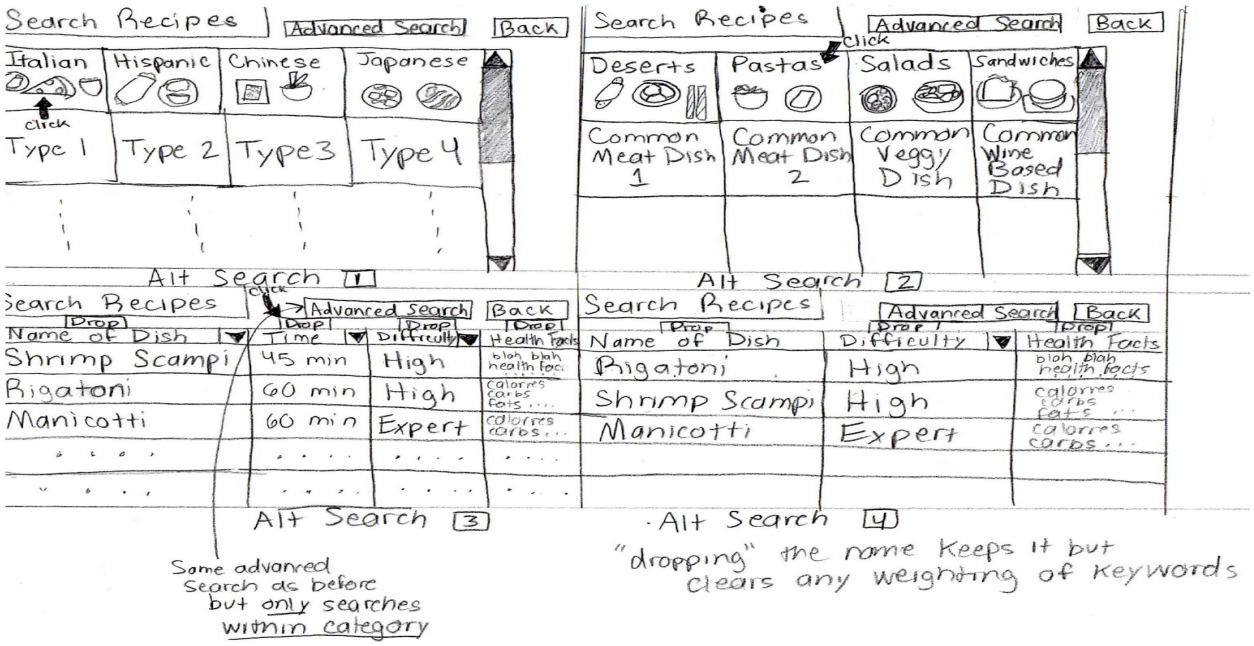
Done

Search

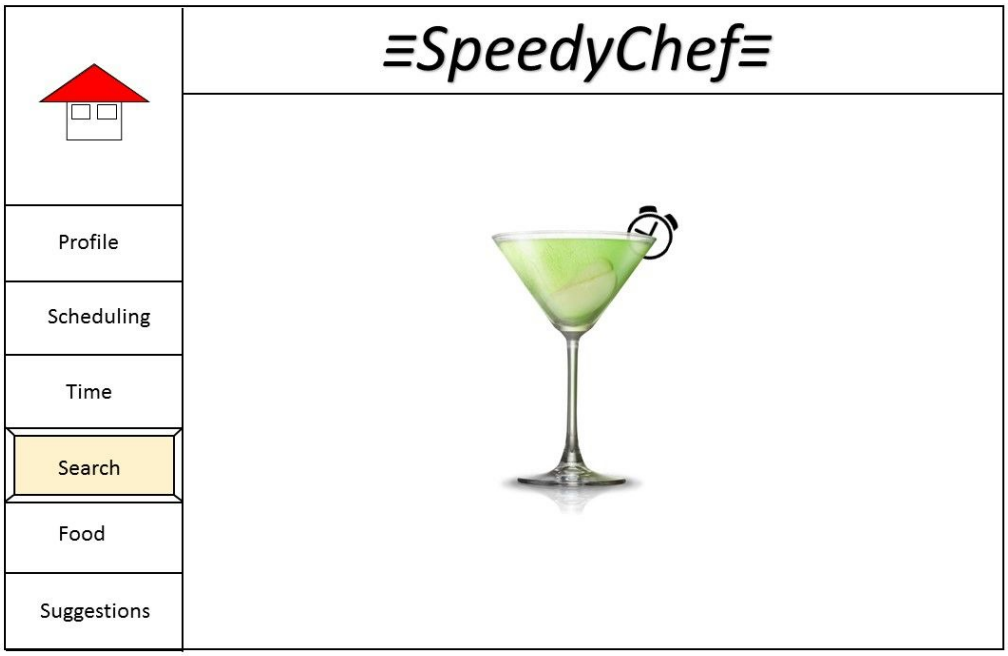
Prototype One



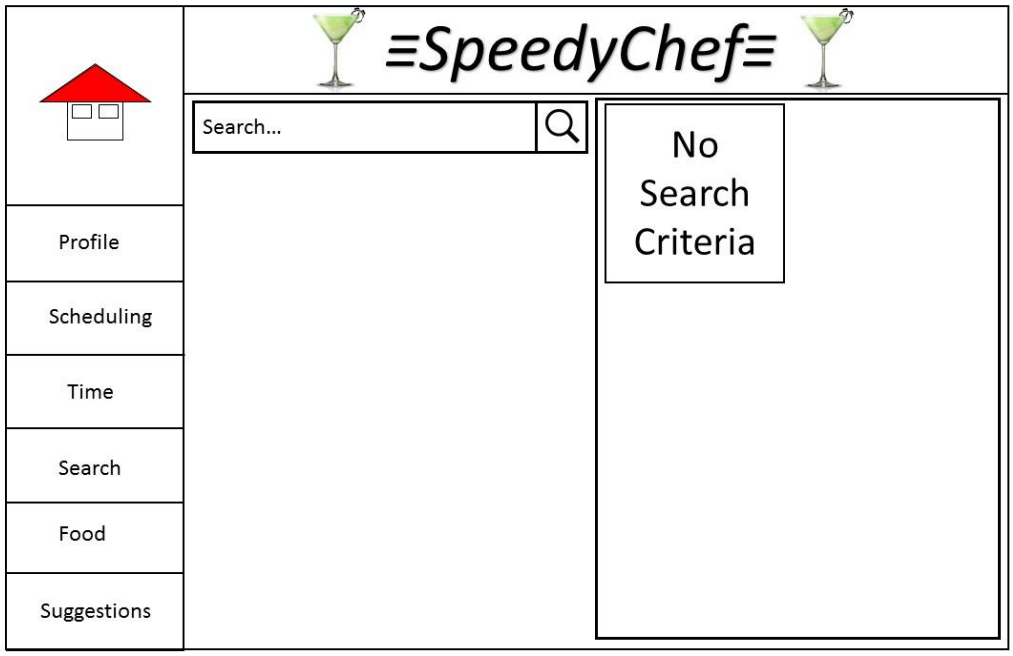
Prototype Two



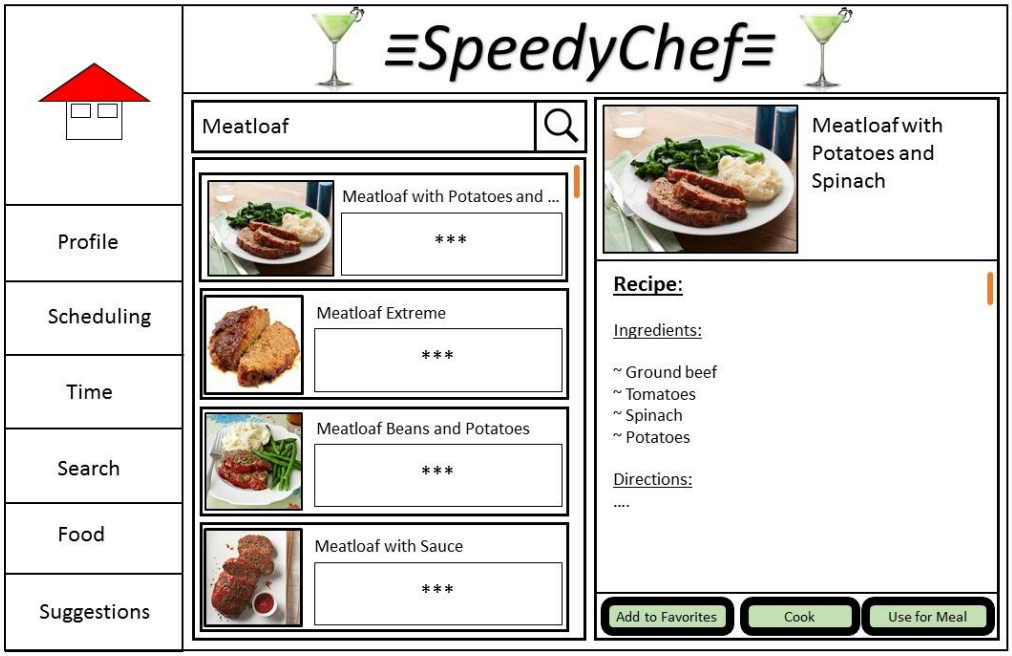
Prototype 3



The user starts at the home screen on his or her tablet. The user then uses his or her finger or stylus to select *Search* option, which provides feedback as shown. To search for food.



A search bar appears, where data loads dynamically. The user can see meal information load while typing. To start searching, click on the search bar and start typing. Lets say the user starts typing in meatloaf. You also get to this page from the calendar *Search* button.



By the user typing in meatloaf, a group of options appear in the choice option, allowing scrolling with a finger. The first option appears in the recipe description area. If the user selects another meal, then the food will appear in that display area. The options for *Add to Favorites* will add the recipe to a favorites list. The *Use for Meal* will allow you to schedule the meal. *Cook* will allow you to do cooking step through for meals. The user then decides to look at the *Suggestion* area, which is based on favorites and search histories.

*** is to represent the a small snippet of the recipe for display purposes. Since I don't know the details, it would have not been in my best interest to look up these detail when I still had other prototypes to do.

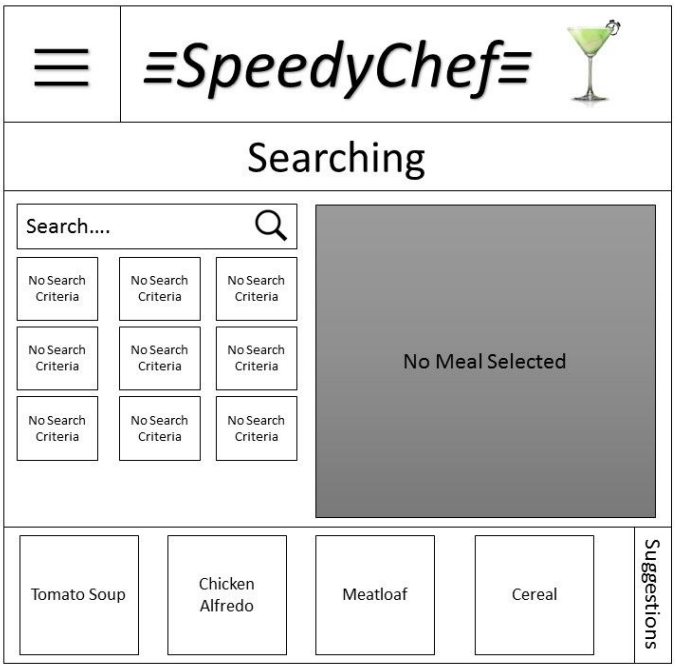
By using the suggestion algorithm described in the previous drawing, a group of options appear in the choice option, allowing scrolling with a finger. The first option appears in the recipe description area. If the user selects another meal, then the food will appear in that display area. The options for *Add to Favorites* will add the recipe to a favorites list. The *Use for Meal* will allow you to schedule the meal. *Cook* will allow you to do cooking step through for meals. The user then decides to look at the *Suggestion* area, which is based on favorites and search histories.

*** is to represent the a small snippet of the recipe for display purposes. Since I don't know the details, it would have not been in my best interest to look up these detail when I still had other prototypes to do.

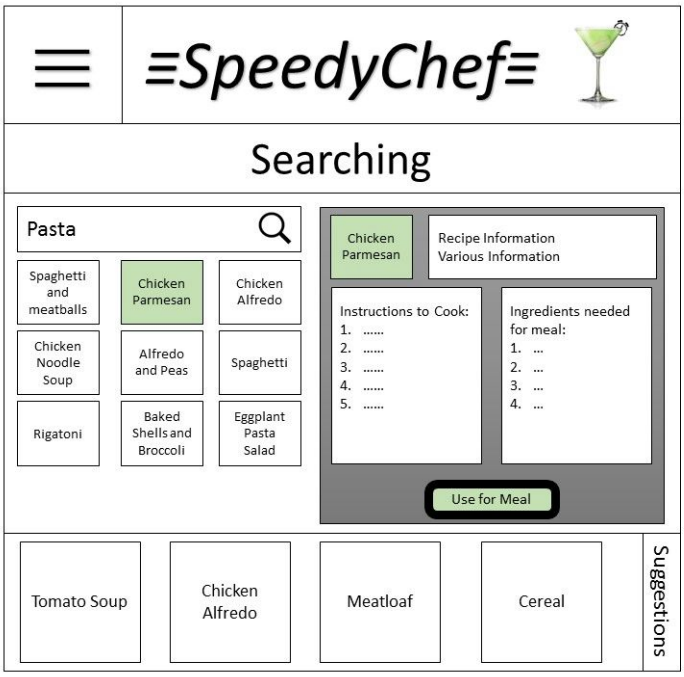
Prototype 4

This is the home screen of the application. The application has the menu button on the left. The user clicks the menu button.

From here, the user can click on the button to take the user to the section that the user clicked. Here, the user clicks on *Search*.



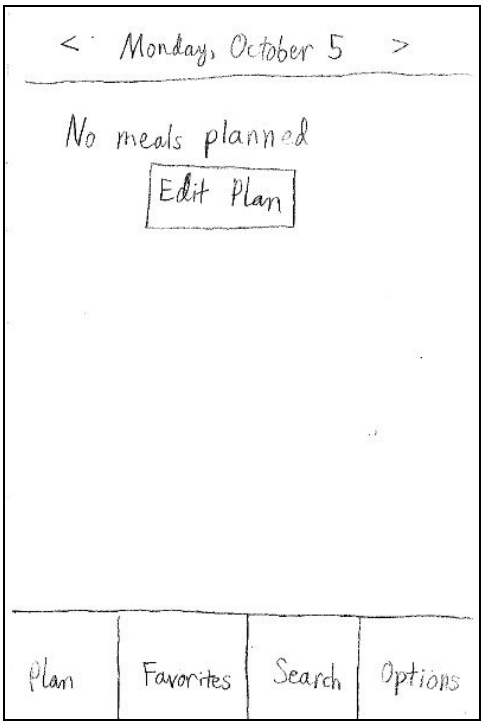
In the search, there is a banner at the bottom with suggestions, which are just pictures of meals. To get more information, the meals can be clicked on and appear in the meal description box. The search information is typing into the box will load suggestions dynamically. By clicking on a meal, information will be loaded into the gray box. As the data dynamically loads, pictures will be displayed instead of words. The words will appear in the gray box when selecting a meal. * For the prototype, pictures were not included. This however brings up a good point of having a back-up if images are not loading, such as a alternative display box of text if the images are not loading.

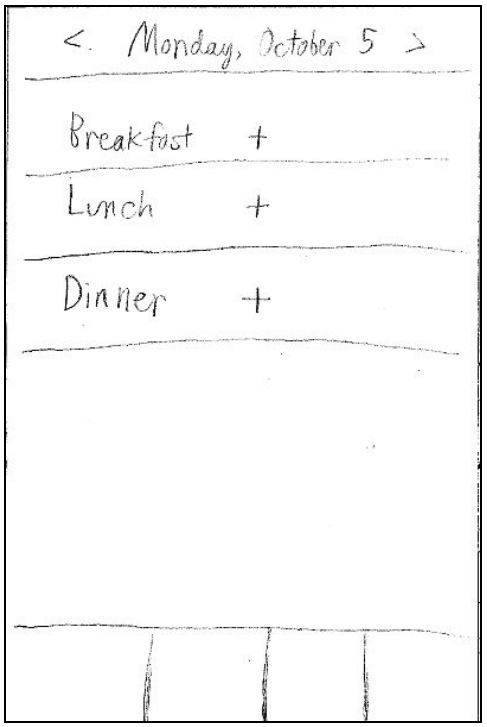


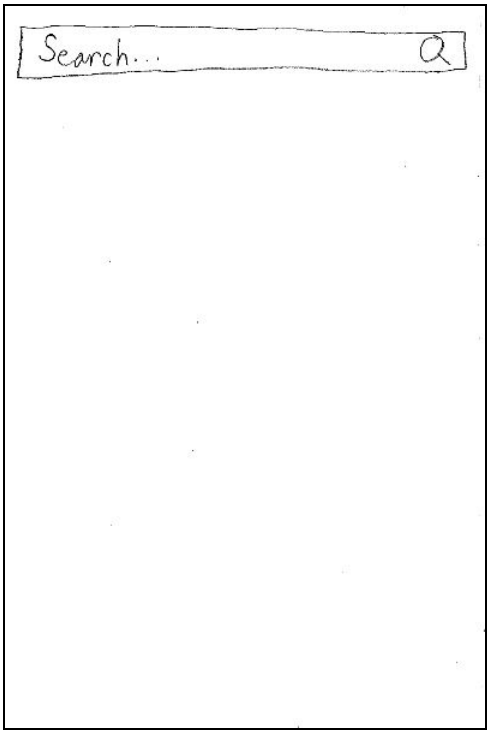
When selecting a meal, information is loaded into the box for the user to read. At the bottom of the box is a button that allows the user to put the meal to a day or for when the user is searching from the schedule page. For this prototype, images were not included for simplicity and details in meals were not included. The suggestions can also be clicked on, which will load the same information into the meal loading area. Suggestions are based on previous searches and other information found in the profile.

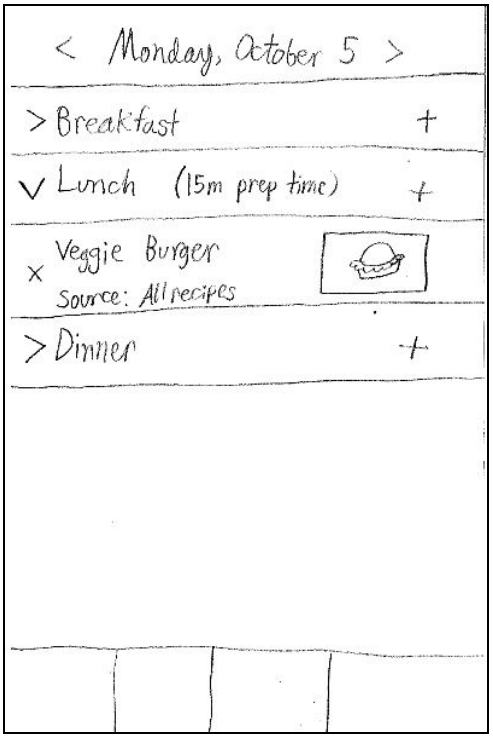
Meal Planner

Prototype One

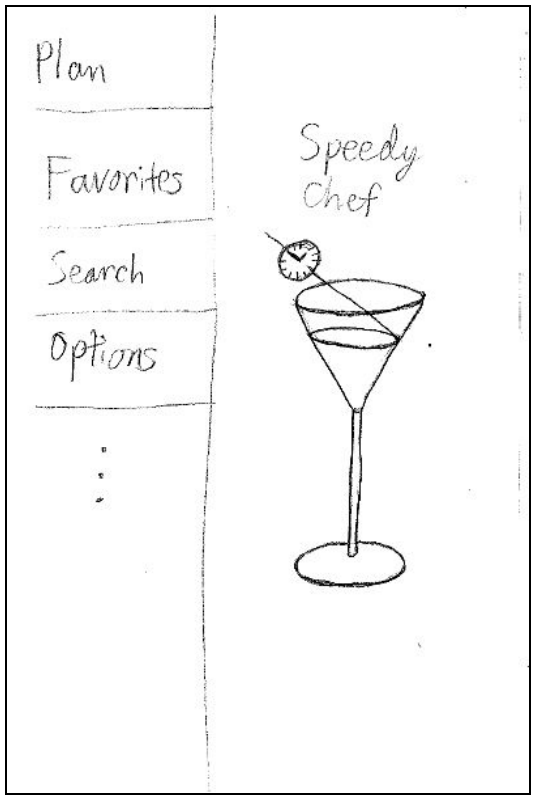


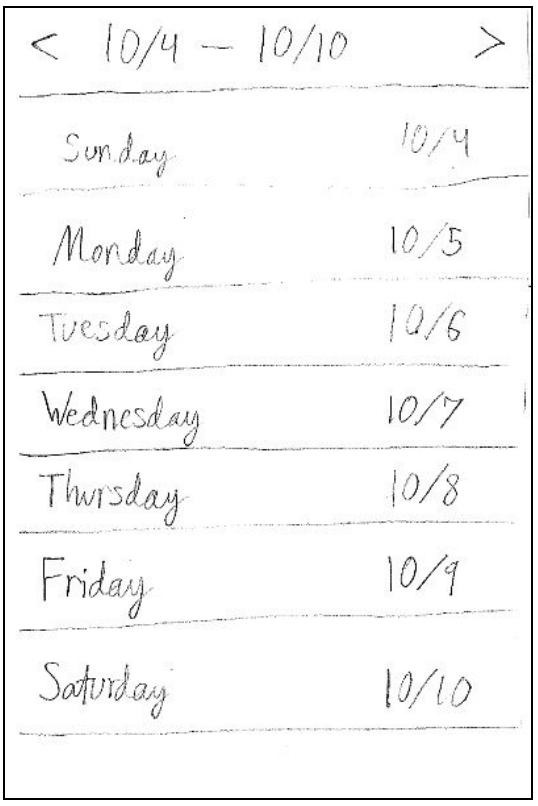


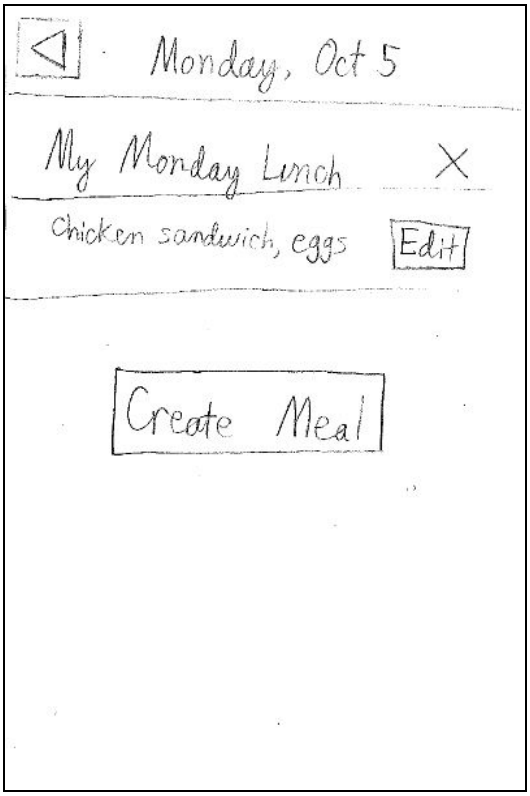


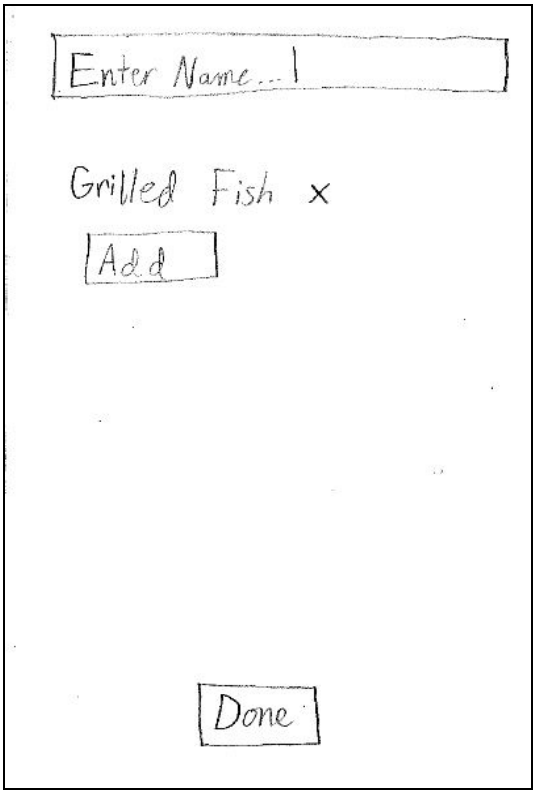


Prototype Two

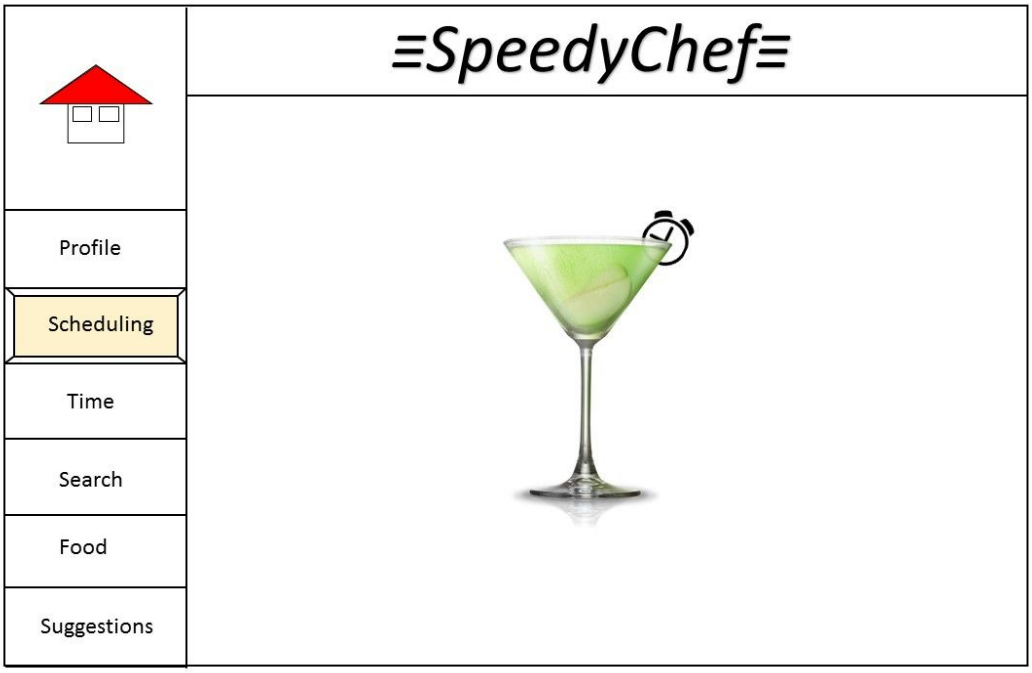




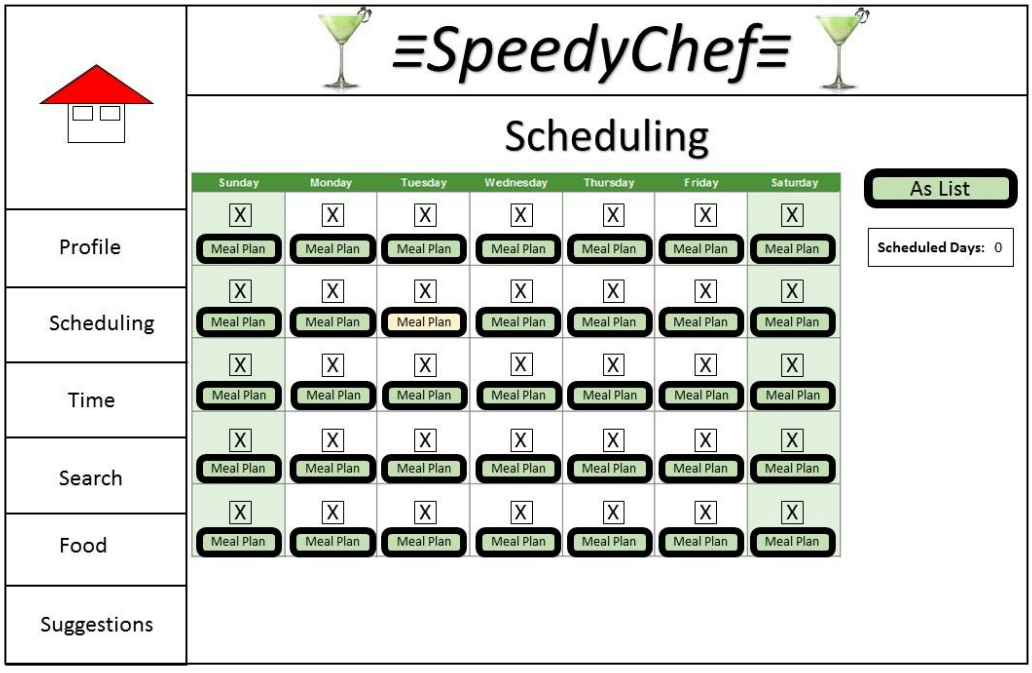




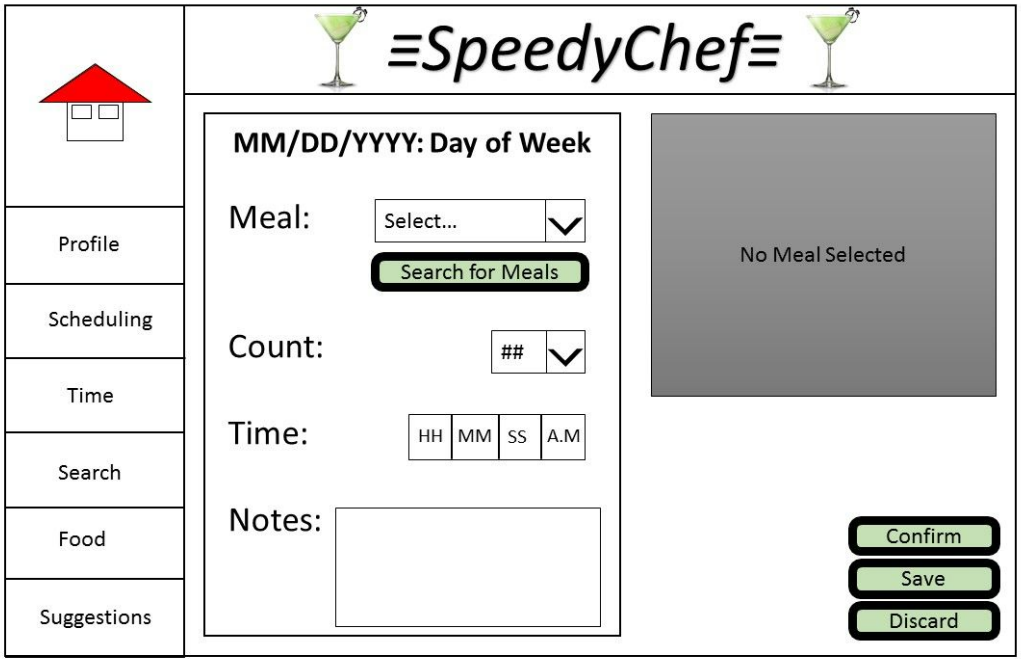
Prototype Three



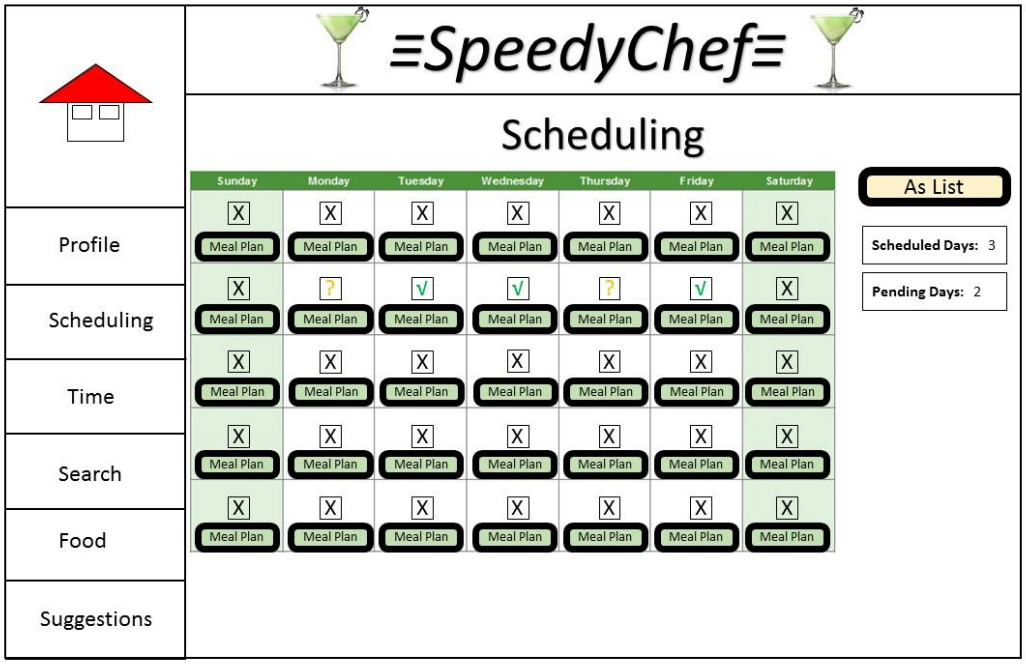
The user starts at the home screen on his or her tablet. The user then uses his or her finger or stylus to select *Scheduling* option, which provides feedback as shown.



The user then comes to a screen where he or she see the available days for the current month. From there, a box on the far right lists the number of scheduled days. On the days not scheduled, it displays an X as a way to show that the day has no meal plan. To schedule a meal for a day, the user clicks on the *Meal Plan* button.



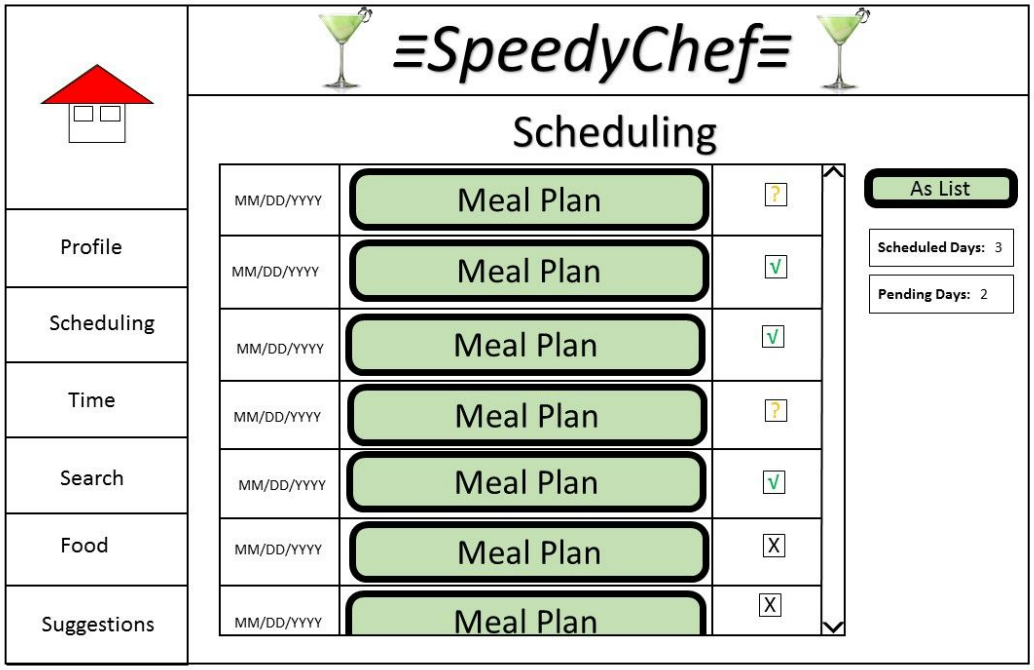
The user then comes to this screen to schedule a meal for a day. Here, the user has the option to drop down from previous selected meals or favorites, or the option to search. The search prototype is seen later. A count for the number of members being at dinner is available to help do servings and ingredient count for the meal. A time can be set for dinner, as a way of notification. The notes section is various information the user wants to put in. The gray box will show a stock photo of the proposed meal being cooked, as a general idea of what it should look like. The option buttons given either do the following, confirm the date and information, save and return to the calendar page if need to fill out later, and discard does not save any changes to the date.



The following shows a couple of iterations of scheduling meals for days, some confirmed and some not completed. Complete days are filled out in green check marks and orange question marks are used for pending days.

The user then clicks on *As List* to remove the calendar on the page to present the dates in a list format.

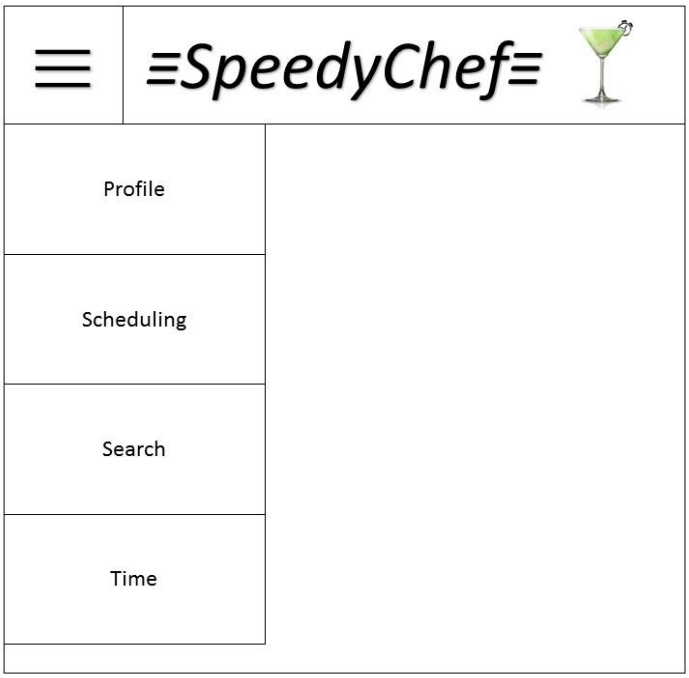
*** Refer to last page for feedback information.



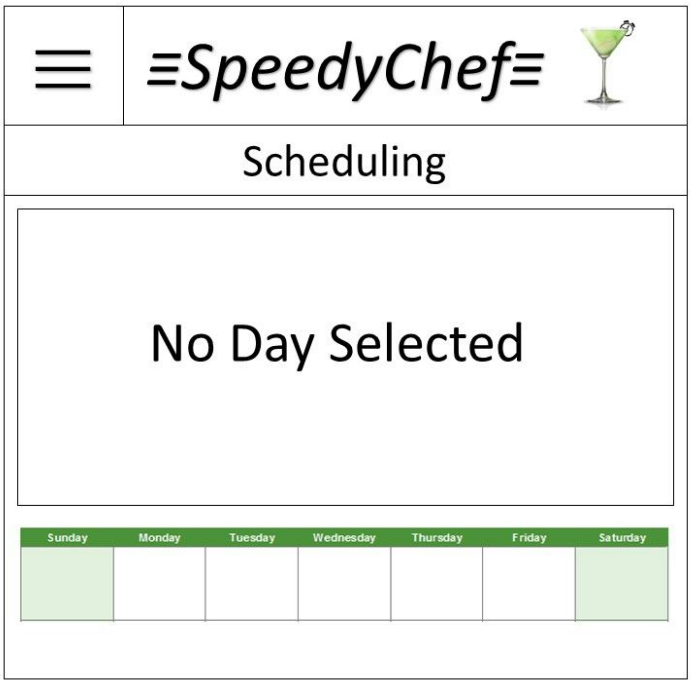
The list representation allows for a scroll bar and see various dates. All of the information is the same, just laid out differently from the calendar view. Same functionality for the user.

Prototype Four

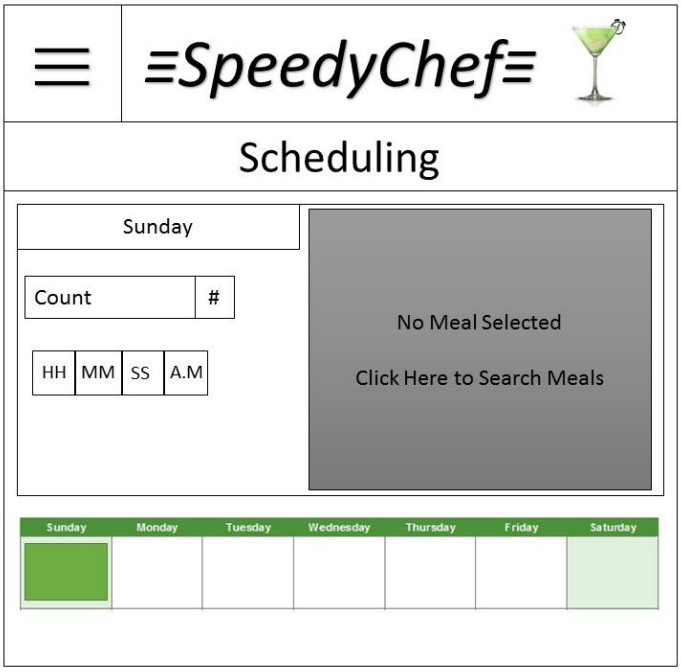
This is the home screen of the application. The application has the menu button on the left. The user clicks the menu button.



From here, the user can click on the button to take the user to the section that the user clicked. Here, the user clicks on *Scheduling*.

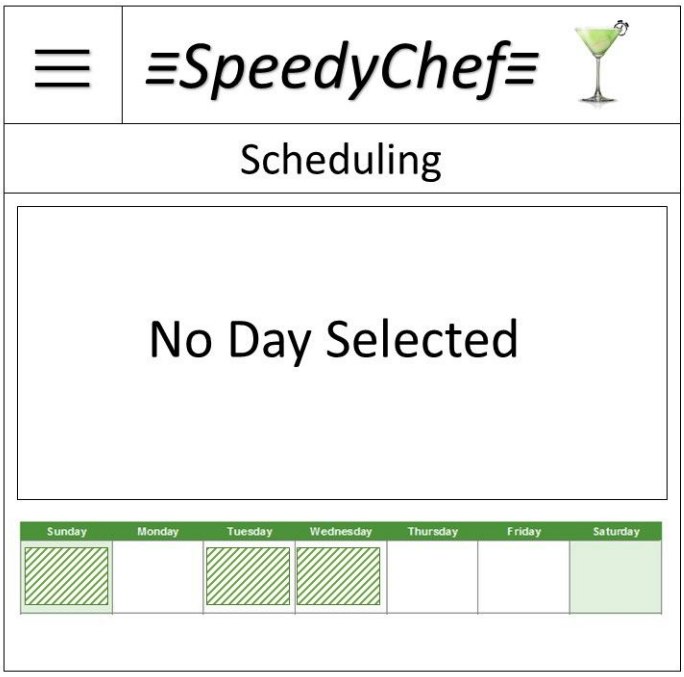


The user is given a week to schedule meals on. To plan meals on a day, the user must click on the day. The box above the calendar is where the user can fill in details of the day selected. The user gets feedback when clicking a day.



The user selects Sunday, and fields are set up to input meal information. To select the meal to cook, the user must click on the picture slot to do searching for meals. From this implementation, the user cannot select from favorites list. The clicking of the picture will take the user to the *Search* screen, which is shown in another prototype.

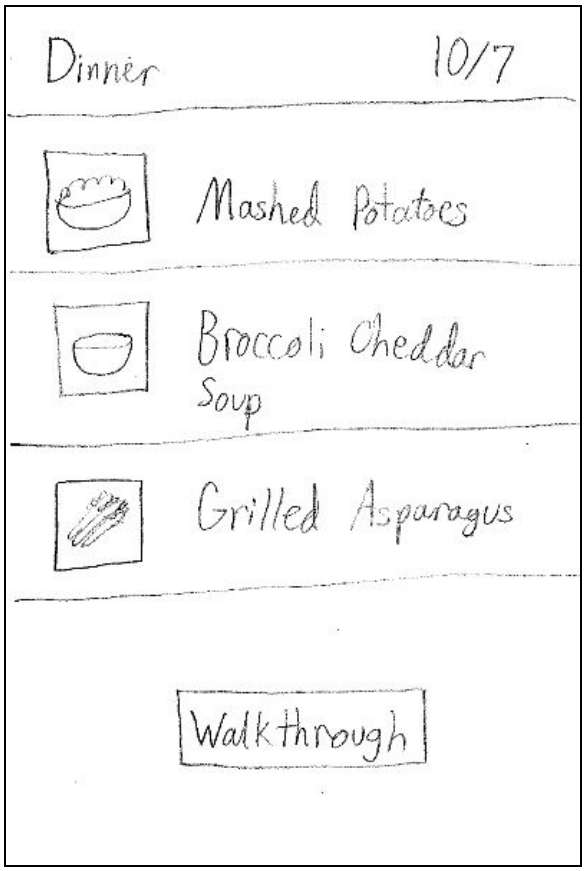
The count field is to show how many members will be accounted for at the dinner. The user can enter a time for dinner to be ready.

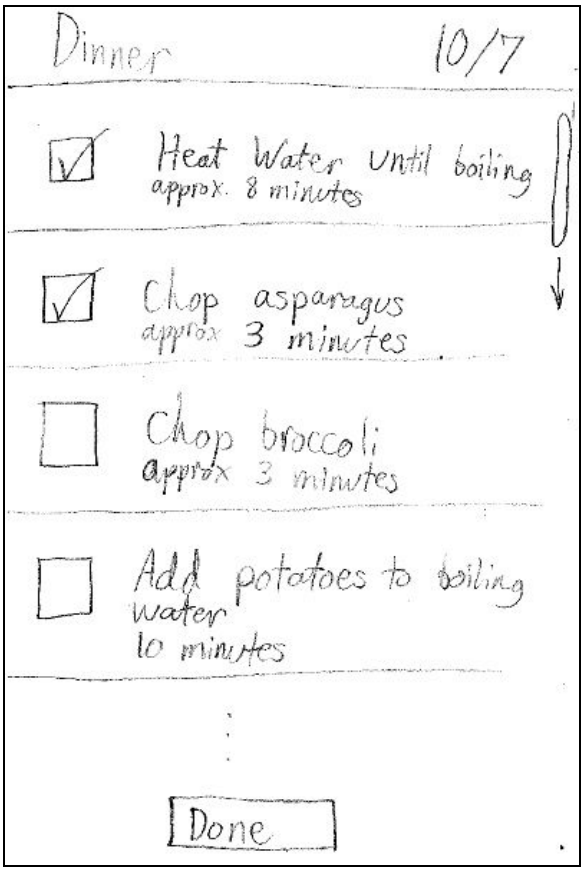


After a couple of iterations, the user has filled in days. The calendar day represent the days that have any information filled in. The selected box will be a solid green box, and filled in information days will be striped pattern. To unselect a day, click on the same box or select another day.

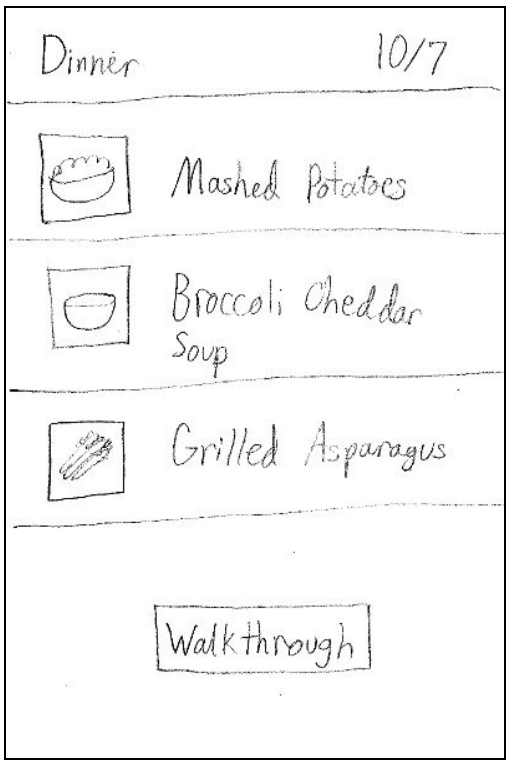
Recipe Walkthrough

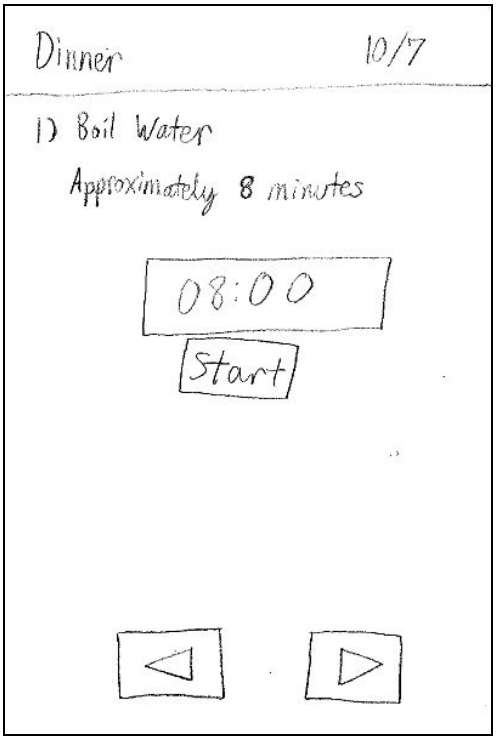
Prototype One

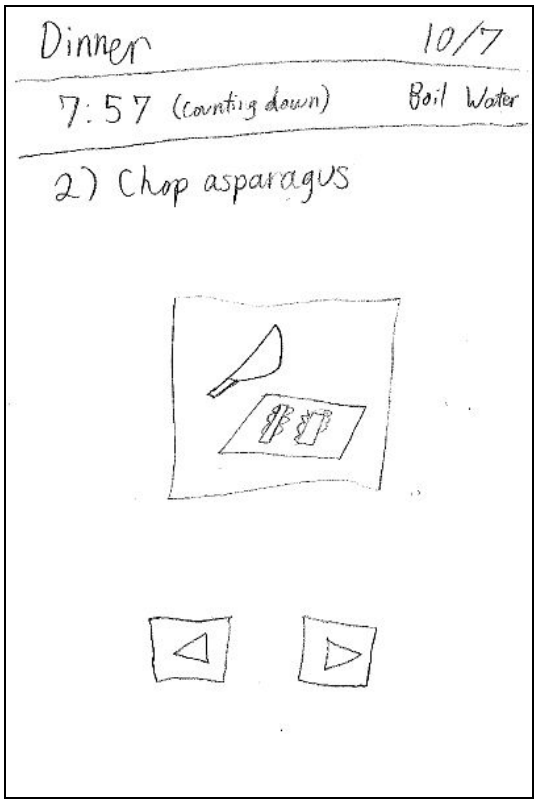


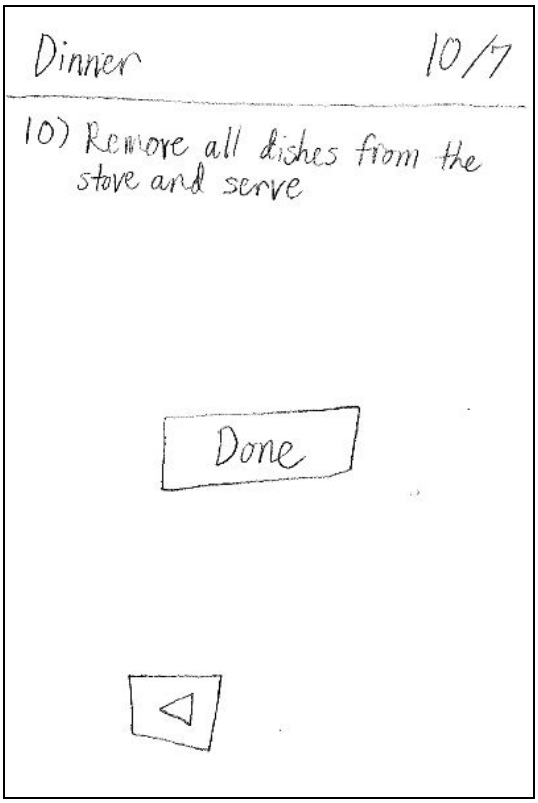


Prototype Two









Prototype Three

Boil vegetables on high heat

00h 30m 00s

Gather Cut Put Add Boil Drain Serve

Drain any remaining water

00h 00m 30s

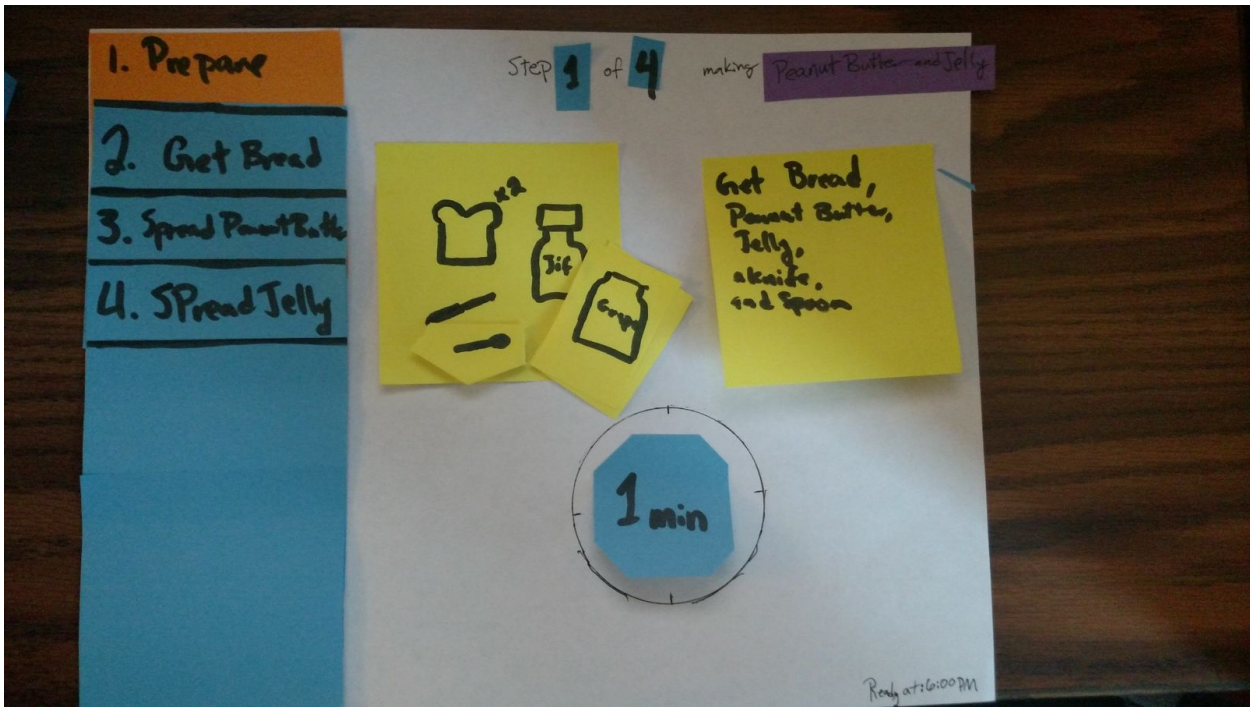
Gather Cut Put Add Boil Drain Serve

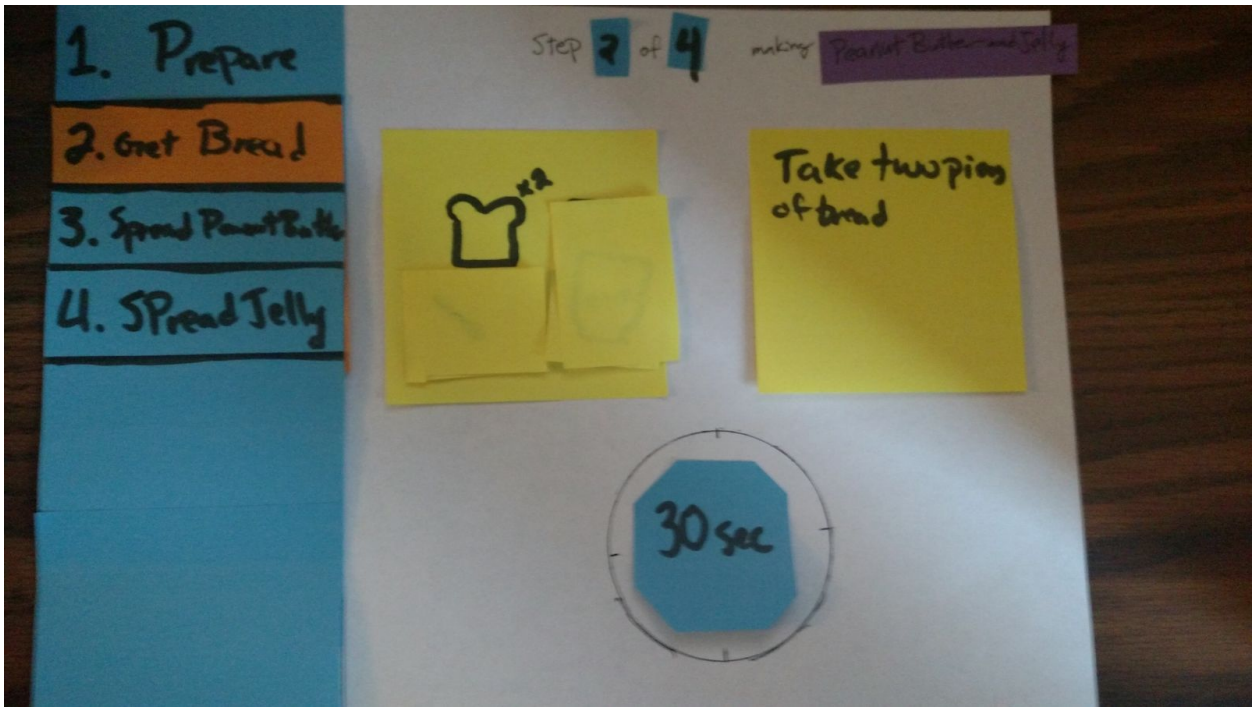
Serve and Enjoy!

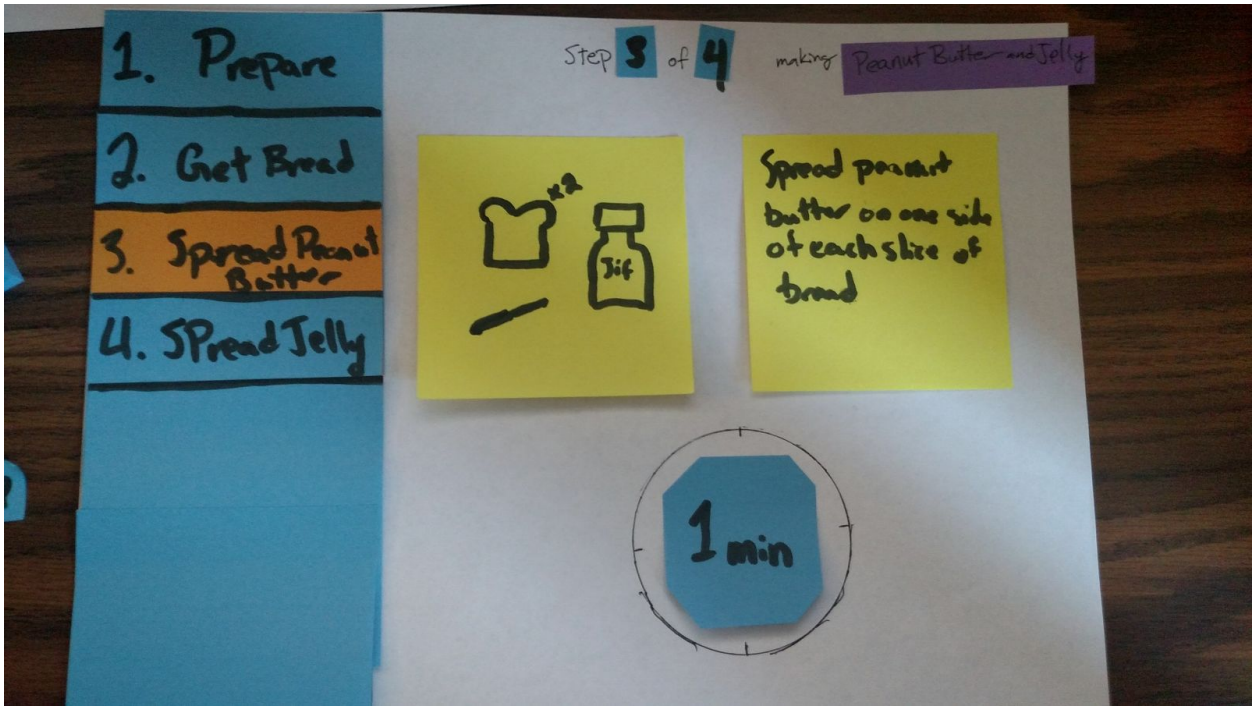
00h 00m 30s

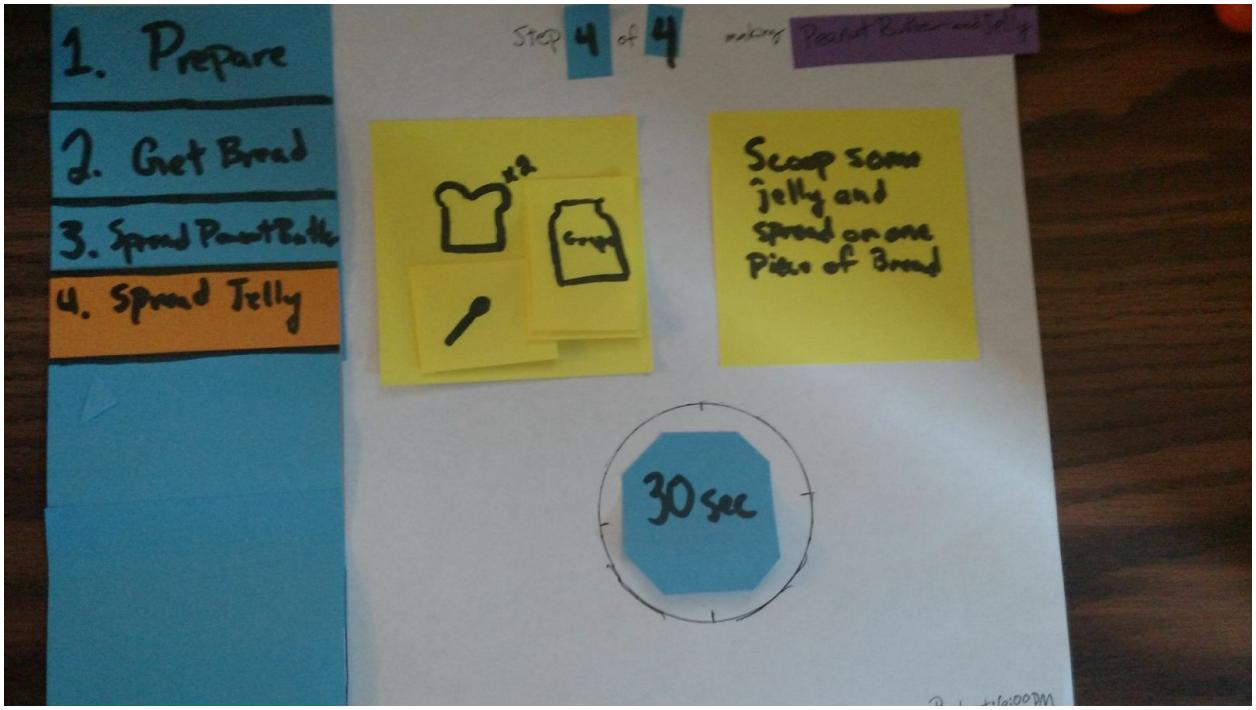
Gather Cut Put Add Boil Drain Serve

Prototype Four






1. Prepare
2. Get Bread
3. Spread Peanut Butter
4. SPread Jelly
Step 3 of 4
making Peanut Butter and Jelly
x2
Jif
Spread peanut butter on one side of each slice of bread
1 min


1. Prepare
2. Get Bread
3. Spread PeanutButter
4. Spread Jelly
Step 4 of 4
x2
Scoop some jelly and spread on one piece of Bread
30 sec

Evaluation Plan

Each of us performed an evaluation for the two features we are responsible for. We showed the users all four prototypes that were created for each feature. All feedback received was recorded and discussed. Based on this feedback, we modeled our updated our design on the popular prototypes while incorporating popular and useful features from the alternative designs. Topics we addressed in our analysis included what features would be most used, how to combine our designs in the end to become one, what would combine to form a fluid prototype, how to eliminate client frustration and increase ergonomics, and how to pick out common difficulties between all designs to address problems we did not anticipate.

Potential Users Interviewed

Logan Erexson - He reviewed Search and Preference features. He lives on-campus, is a chef for himself and his suitemates, and has a very technical background as a computer scientist. He has access to a smartphone and laptop and would prefer to use the smartphone.

Jacob Knispel - He reviewed Search and Preference features. He lives on-campus, is a chef only for himself and does not cook regularly, and has a very technical background as a computer scientist. He has access to a smartphone and laptop, but would prefer to use the smartphone.

Daniel McGarry - He reviewed Search and Preference features. He lives on-campus, is a chef only for himself, and is familiar with technology, especially desktop computers. He has access to smartphone and laptop applications and would prefer to use the smartphone.

Tim Smith - He reviewed Search and Preference features. He lives on-campus, is a chef for himself and occasionally others, and is familiar with technology. He has access to a smartphone and laptop and desktop and would prefer to use laptop or smartphone.

Jack Porter - He reviewed the Meal Planning and Recipe Walkthrough features. He lives on campus but rarely uses his Rose-Hulman meal plan to get prepared food. He cooks very simple meals, usually only for himself. He habitually uses an Android phone.

Rachel Weber - She reviewed the Meal Planning and Recipe Walkthrough features. She lives on campus and cooks from time to time in the Percopo kitchen. She prepares meals only for herself. She is comfortable using a smartphone.

Lauren Fischer - She reviewed the Search and Meal Planning. She lives on campus and cooks many of her dinners in her room. She prepares some meals with her boyfriend. She can utilize a smartphone.

Jacob Tyler - He reviewed the Search and Meal Planning. He lives on campus and does a small amount of cooking for himself, mainly uses the dining services. He knows how to use his smartphone very well.

Results

Preferences

Of the two choices that were laid out by Steve, the one that really stood out as an obvious winner was the self-contained preferences tab with subtabs/subsections. The interviewees seemed attracted to the efficiency of the advanced search and save method of the alternate design, but ultimately found it a bit tedious and prefered a more direct route to see their preferences/settings. Across the board, more meant less; comments like “Do any tools really need this many options?” and “Clunky” and “Cluttered” told me that simplicity is the key, even with a very input-centric feature such as preference submission. Unclear wording was also a plague with comments like “unclear what this does” and “not clear what these do” (referring to the Commit and Return button and Drop buttons respectively for Alt Preferences 3 and Search 3). Generally, then, it seems that pictures and careful phrasing and placement are key; it would be best to put “X”s and arrows in many places where worded buttons were once placed. We decided after much discussion that the swipe feature combined with indicators of what pane you are on could declutter and unbutton as well as provide more real estate for display and enlargement of prompts.

Search

Unanimously, search was seen as a feature that should perpetually be available to the user. Either in a drop down menu or as an input bar, there should always be a route to searching - especially the idea of a home screen search bar front and center. The differentiation between a browse feature and a traditional search feature was made apparent by comments like “Maybe add this as a browse option.” Many users desired to have both be made available and not be forced to have one or the other only. We decide to merge these and have a quick browse that leads to a common search screen. We believe this will add the best of both worlds in the least number of clicks and possible screens so that users can acclimate and make best use of our most important feature sooner. Pictures were something that “seems easier to glance through,” which was a recurring theme across features for most people. Advanced search was unanimously seen as unnecessary. People preferred to have preferences that influenced the search behind the scenes while having the most important filters, keywords, time, and skill level, able to toggle on demand during the search. Dynamic typing and searching was a must across the board as well; in preferences as well as search it is important to capture and refine the idea that the user does not want to be specific, he or she wants to throw a dart close to the bullseye and have our app take the dart and reposition it using inferences.

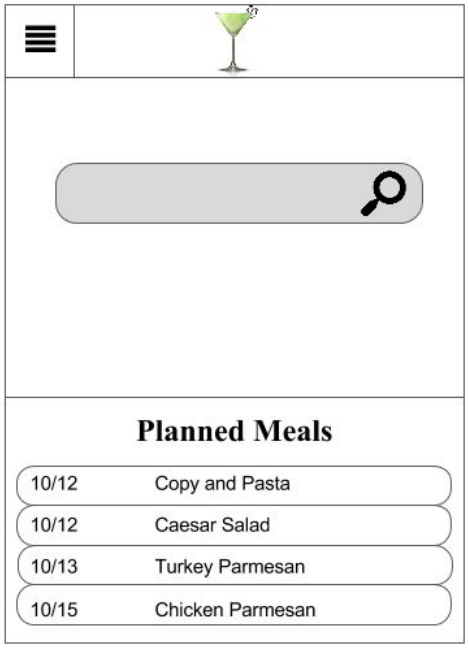
Meal Planning

Many users found the month long view a very “cluttered” screen and was too much to work with. Users prefered having the option to name the meal compared over simply being offered “Breakfast”, “Lunch”, and “Dinner.” The two designs by Alia were prefered overall, with minor combination between the two. Minimizing the number of clicks was a big concern when moving over several days, many users wanted a quick way to shift weeks at a time. For users that had already searched recipes and added bookmarks, adding a meal from that list is simple. Users should also be able to add a recipe obtained through the search or browse feature.

Recipe Walkthrough

Users were opposed to the “list” design of Prototype 1. They felt that displaying the recipe steps individually could make more information and functionality available. They wanted the step display screens to provide some indicator of how far along the user is in the recipe. Opinions on the timer feature were mixed. One user liked the inclusion of the timer, and she wants this feature to be included for all steps, even quick and simple ones. She thinks that the timer would motivate her to accomplish tasks quicker. Another user disliked the timer feature. He said that he would not use it because there are other timers available in his kitchen, such as on the oven or microwave. He preferred the aesthetic of the “clock” icon that displays the time in Prototype 3. Additionally, one user liked the inclusion of a picture for each step, while another thought it made the GUI design seem childish.

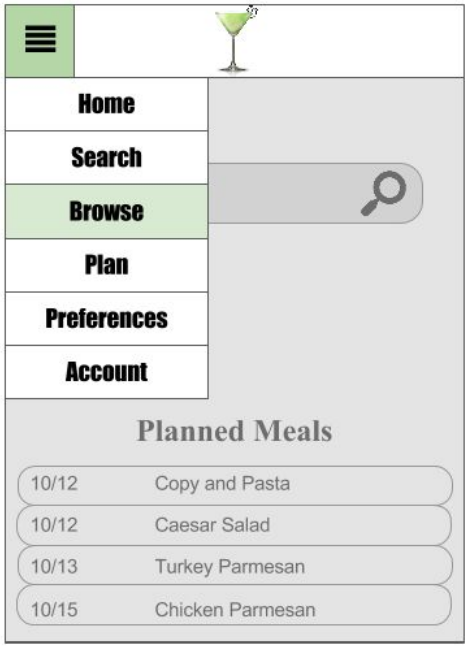
Revised Prototype



Home Screen

This is the start screen view for the SpeedyChef. The view has a search bar to click on and will take you to the **SEARCH (1)** page once information has been typed in and the search button or enter button has been clicked. The bottom of the screen has a current calendar showing the planned meals so far.

2

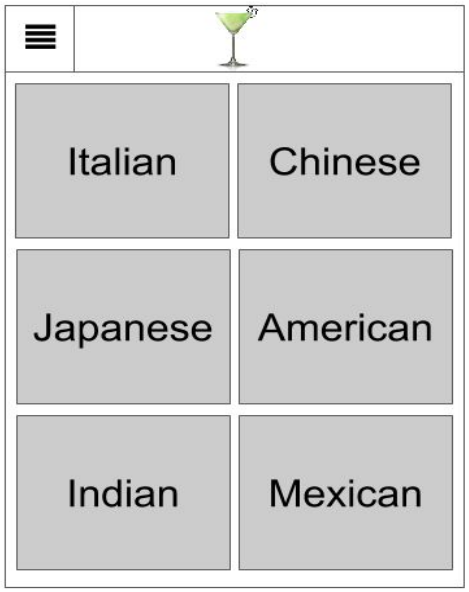


Menu Open

The view when the menu button (≡) is clicked on. Brings up this the 6 options to go to. When the menu button was clicked, everything that is behind the menu becauses faded, showing where the focus is for the application.

From here the user clicks on the ***BROWSE*** **(1)** menu option.

3

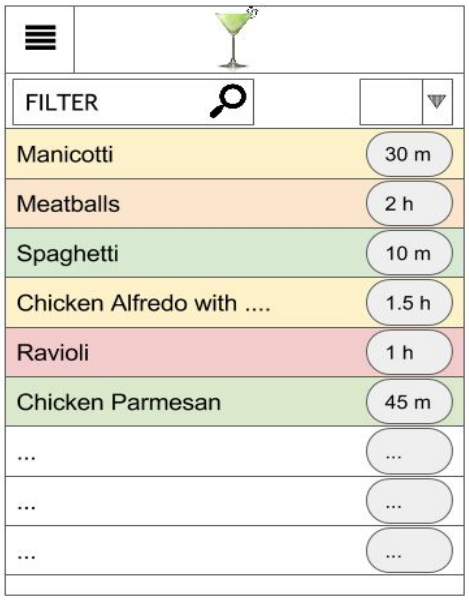


Browse (1)

Once the *BROWSE* button in the menu is clicked, this view pops up cuisine options. This view is scrollable, showing additional options. From here, the user clicks on *Italian*.

Browse (2)

Once clicking on *Italian*, the user is shown all of the subcategories underneath Italian. Here, the user can click on the type of meal for the cuisine. Here the user clicks on *Pastas*.

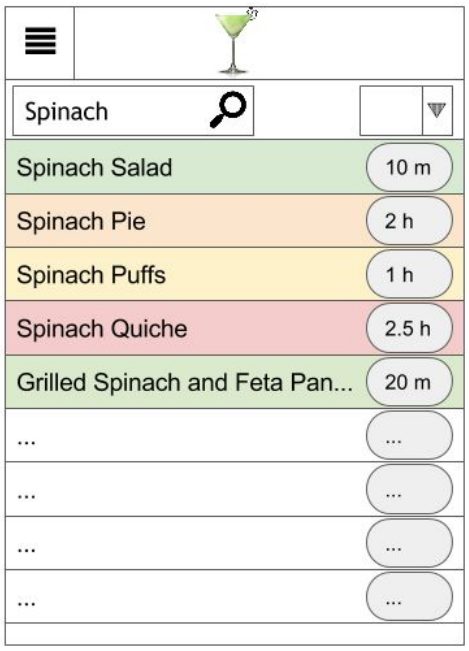


Browse (3)

After selecting the subcategory, a list of options is given. In this view, you have the option to add more *FILTER* keywords, and able to *SORT* the results by various fields. Initially, there are no *FILTERS* and *SORT* options. In the results, which loads dynamically, the whole banner the information is presented on is clickable, taking you to a **VIEW** page.

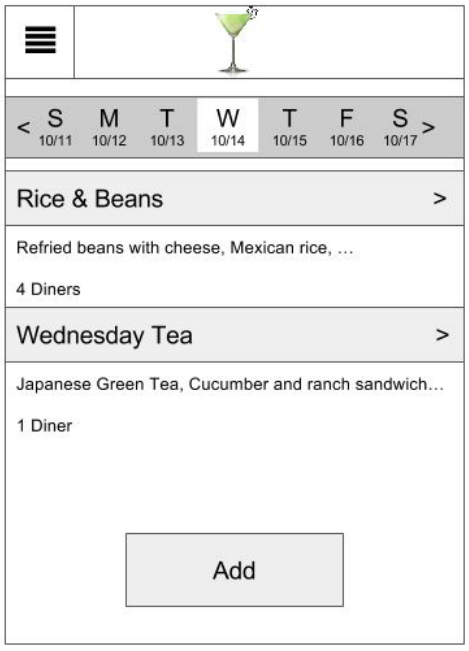
View

Once a meal is selected to **VIEW**, this screen appears. A picture of the meal is shown, and if the picture will not load, alternative text will take the picture's place. A user can favorite a meal, turning the star from white to gold, indicating a favorite for the user, which can be used with the meal planner. If the user is on this screen during scheduling, the user can click *Add to Meal* button to add this meal to the current to the schedule. The *Start* button will allow the user to start a **WALKTHROUGH** of the instructions. Below the picture is the name of the recipe, followed by a section where a user can expand the *Nutrition* values and information, followed by the spot to expand the *Ingredients* section, which is scrollable inside of the area in the view. The *Step* section will keep expanding to the bottom of the page, making the whole **VIEW** page scrollable.



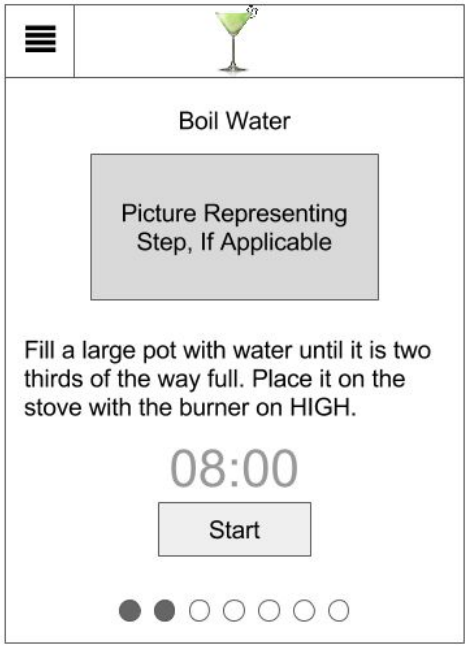
Search (1)

Alternatively, users may use the **SEARCH** feature to navigate directly to this page. This is formatted the same way as the third screen of the **BROWSE** feature, but it is not initially populated with data. Users can search for recipes based on keywords and sort the results by relevance, by name, by estimated completion time, or by difficulty. The colors indicate the difficulty of a recipe; green indicates an easy recipe while red indicates a difficult one. Clicking a recipe takes the user to the recipe **VIEW** page.



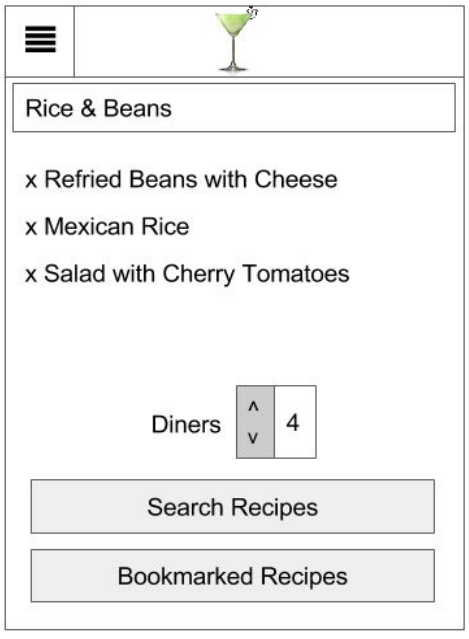
Meal Planning (1)

The **PLAN** page displays all of the meals that the user has scheduled for a given day. The days of the week are displayed at the top, with the current date being selected by default. Users can therefore scroll through multiple weeks at a time and easily select their desired date. The page displays the name of the meal, the planned dishes, and the number of diners. Clicking on the arrow takes the user to a screen where they may edit the meal's details.



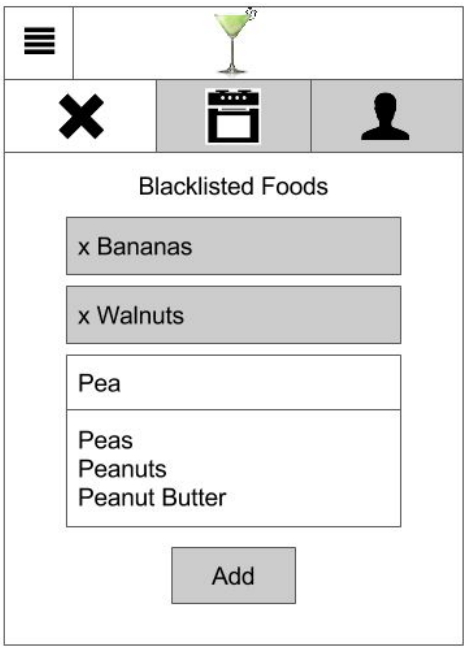
Recipe Walkthrough (1)

The recipe walkthrough feature compiles all of the tasks necessary to complete a meal into a list. The user may swipe left and right to progress in this list, or they may select from the circles at the bottom representing each step. In this example, “Boil Water” is the second step. Pressing the “Start” button causes the timer to begin counting down.



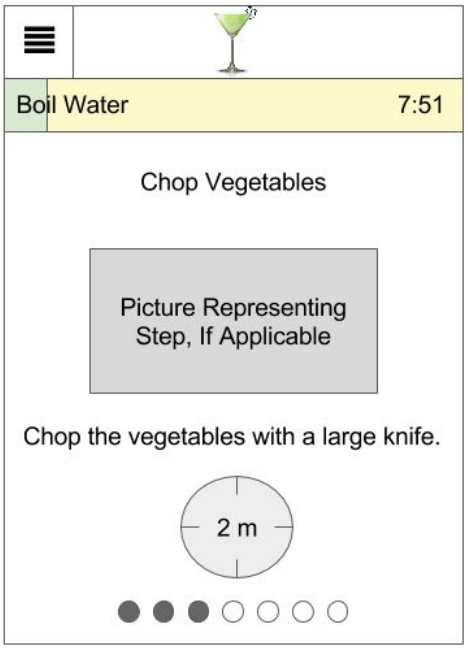
Meal Planning (2)

The plan editing page allows the user to input a name, add recipes to their meal, and select the number of diners. The “Search Recipes” button takes the user to the **SEARCH** page. When viewing a recipe they find through this search, they will have an “Add To Meal” option, which adds it to the meal currently being edited. The “Bookmarked Recipes” button will allow users to view a list of bookmarked recipes from which they may select a dish.



Preferences(1)

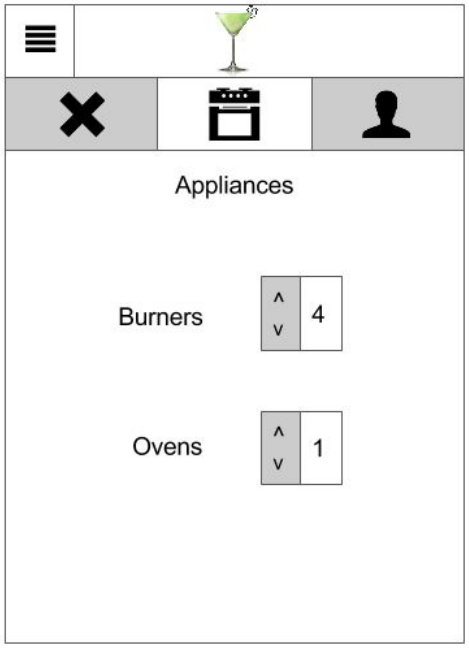
The **PREFERENCES** screen is split into multiple tabs. The “Blacklisted Foods” tab allows the user to specify ingredients that should not be present in any recipes that they cook, typically because the user is allergic. A text field allows a user to input ingredients. As they type, the app should offer suggestions to help them find ingredients.



Recipe Walkthrough (2)

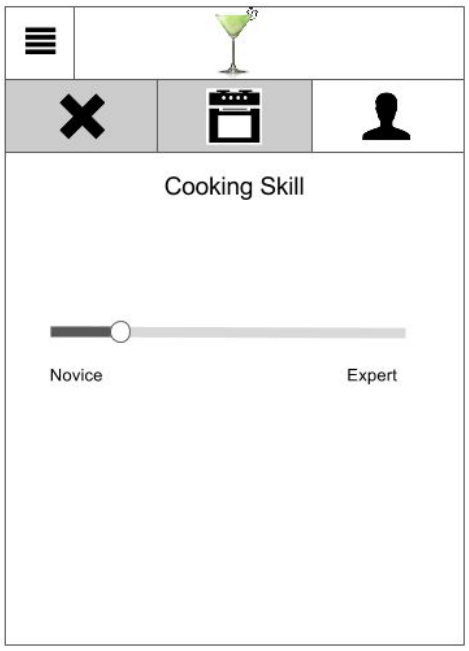
As the user progresses to subsequent steps, the timer is visible at the top of the screen. An animated colored bar progresses from one side of the screen to the other as the timer counts down. This ensures that users know to check on their items (in this case, boiling water) after the time has passed.

The “Chop Vegetables” step does not require a timer. Here, the time is displayed in a clock graphic that should not be confused for a timer.



Preferences(2)

The “Appliances” Tab allows the user to specify what appliances they have available in their kitchen. By default, the app will assume that they have a four burner stove and an oven. The recipe walkthrough feature should not instruct a user to use more burners simultaneously than what they have.

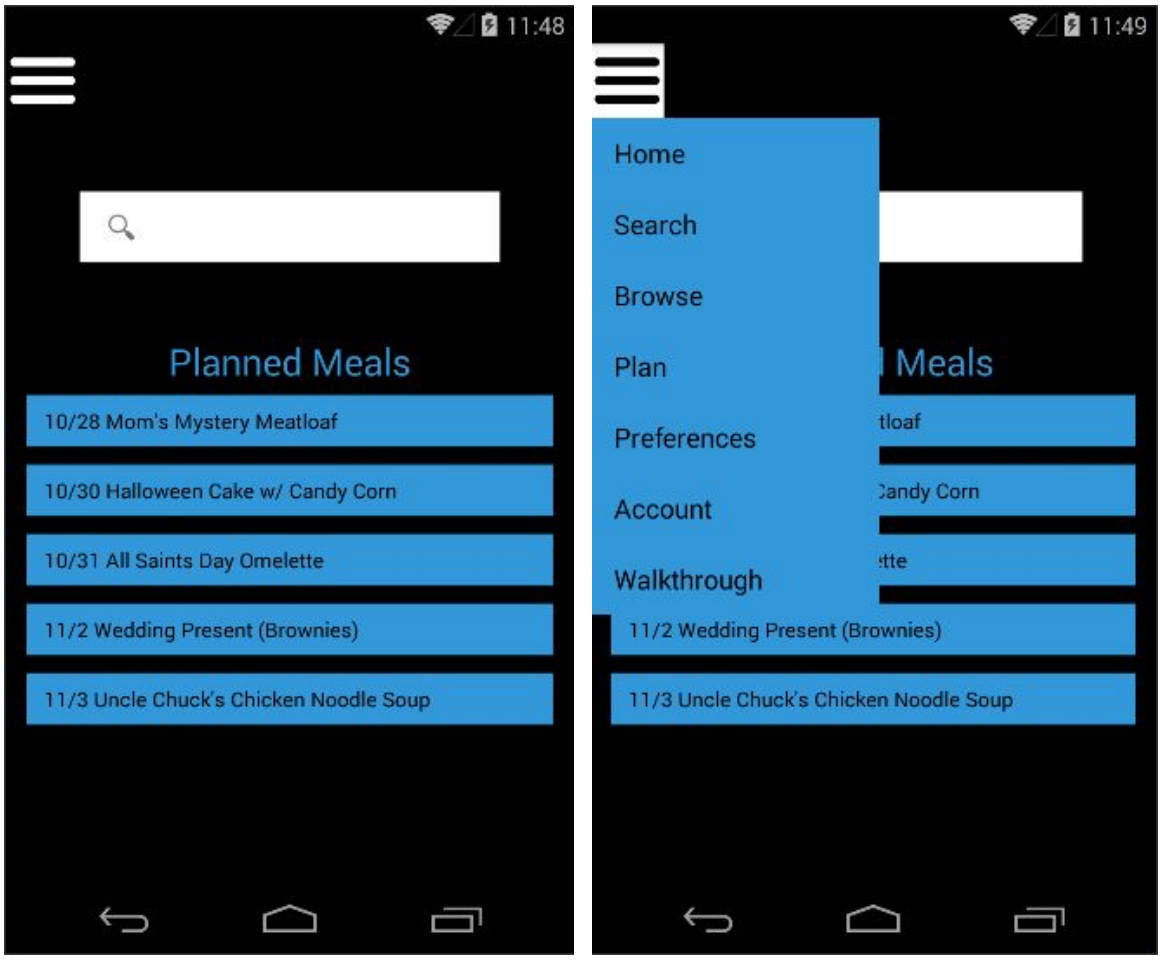


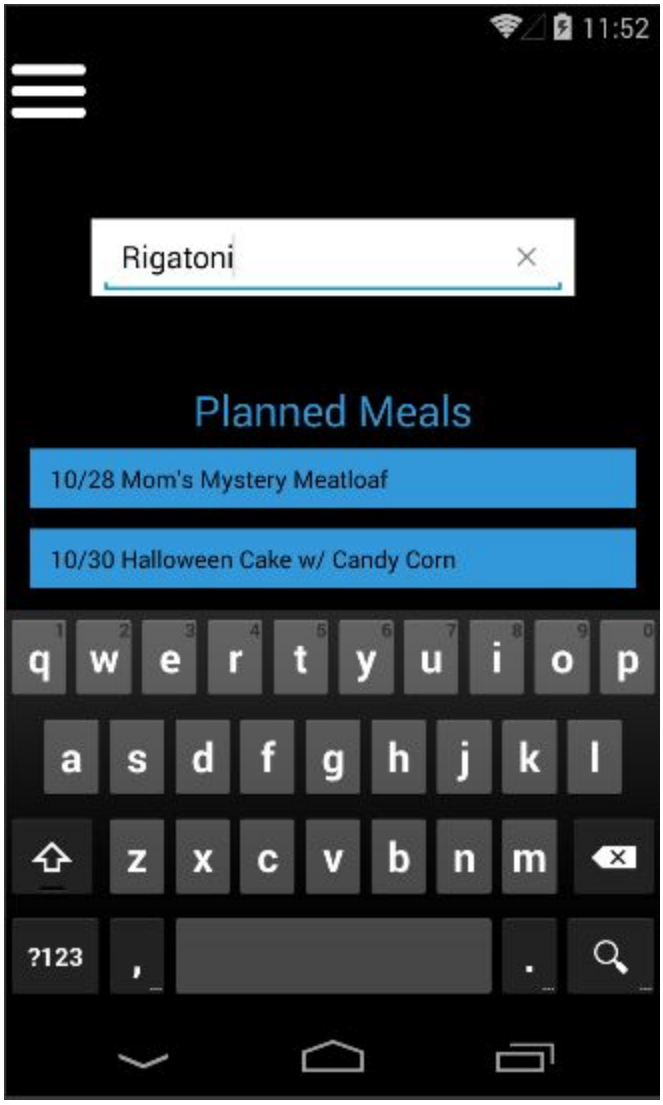
Preferences(3)

The “Cooking Skill” tab allows the user to specify their comfort level with cooking using a simple slider. This defaults to an average setting. This will slightly affect the time taken to perform some tasks, assuming that experienced cooks will perform some tasks more efficiently.

High Fidelity Prototype

Home Screen



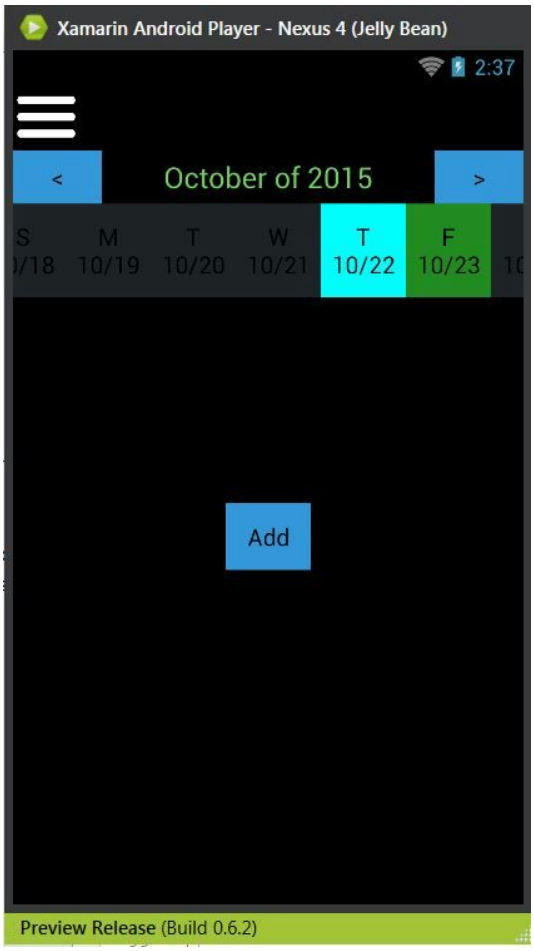

11:52
Rigatoni
Planned Meals
10/28 Mom's Mystery Meatloaf
10/30 Halloween Cake w/ Candy Corn
?123

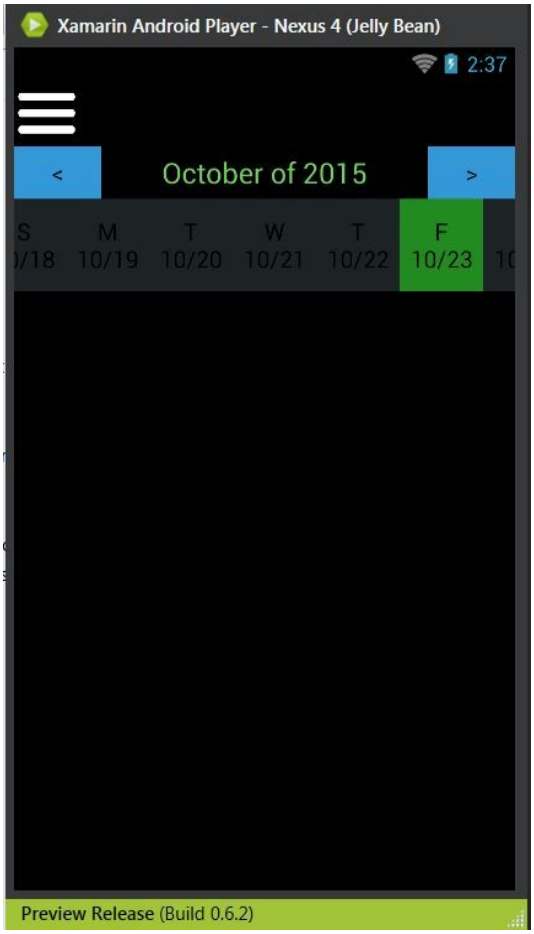
Browse

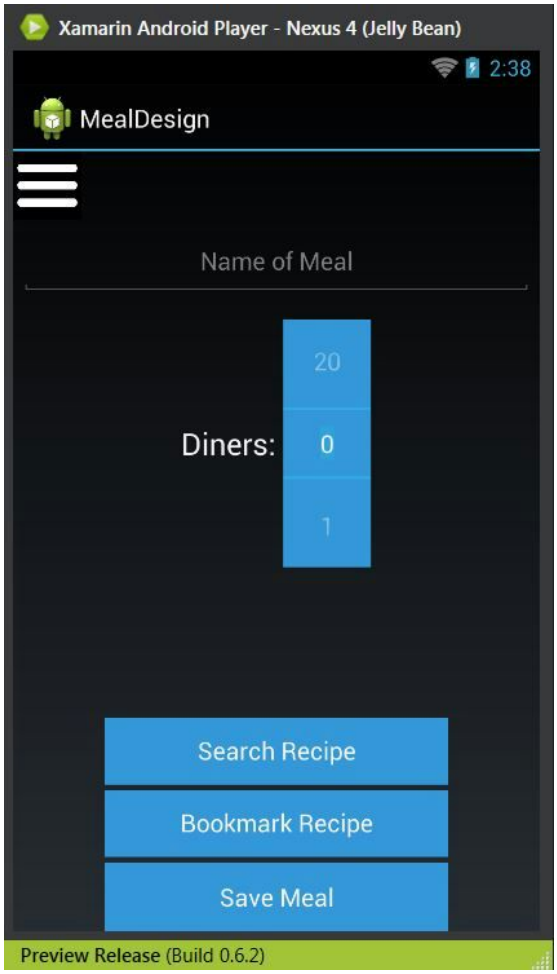

11:47
Italian
American
Chinese
Mexican
Indian
Japanese
French
Thai


11:53
Italian Desserts
Italian Meats
Italian Vegetables
Italian Pastas

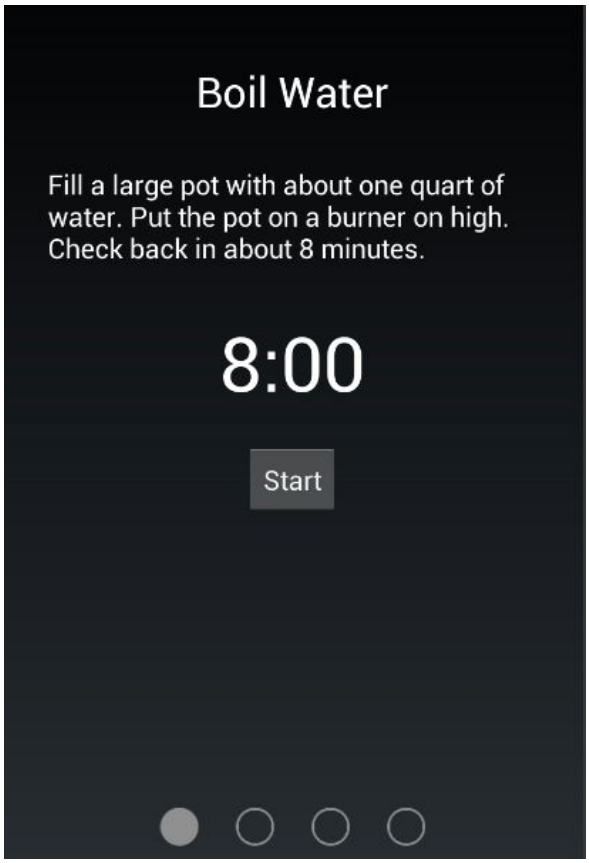
Meal Planner

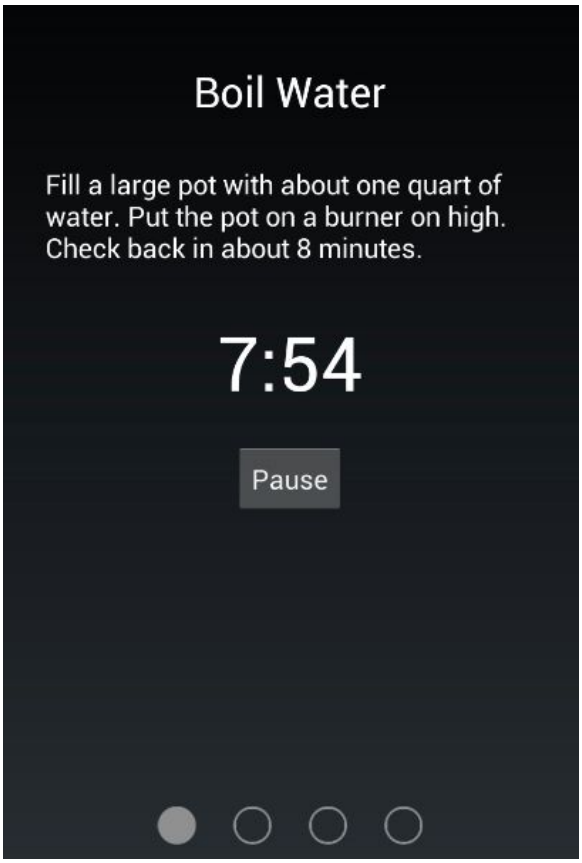





Xamarin Android Player - Nexus 4 (Jelly Bean)
2:38
MealDesign
Name of Meal
Diners:
20
0
1
Search Recipe
Bookmark Recipe
Save Meal
Preview Release (Build 0.6.2)

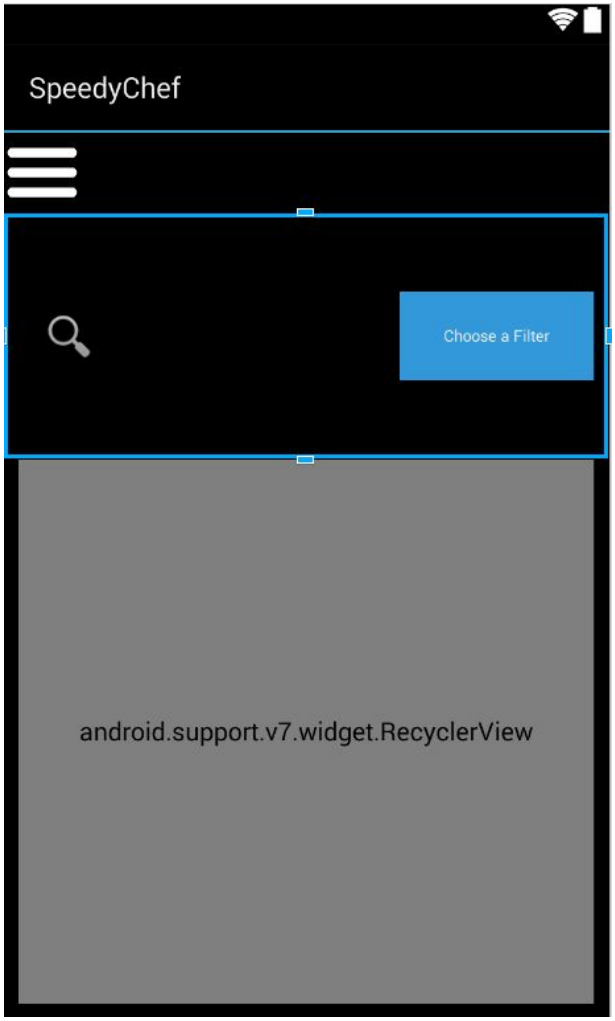
Recipe Walkthrough



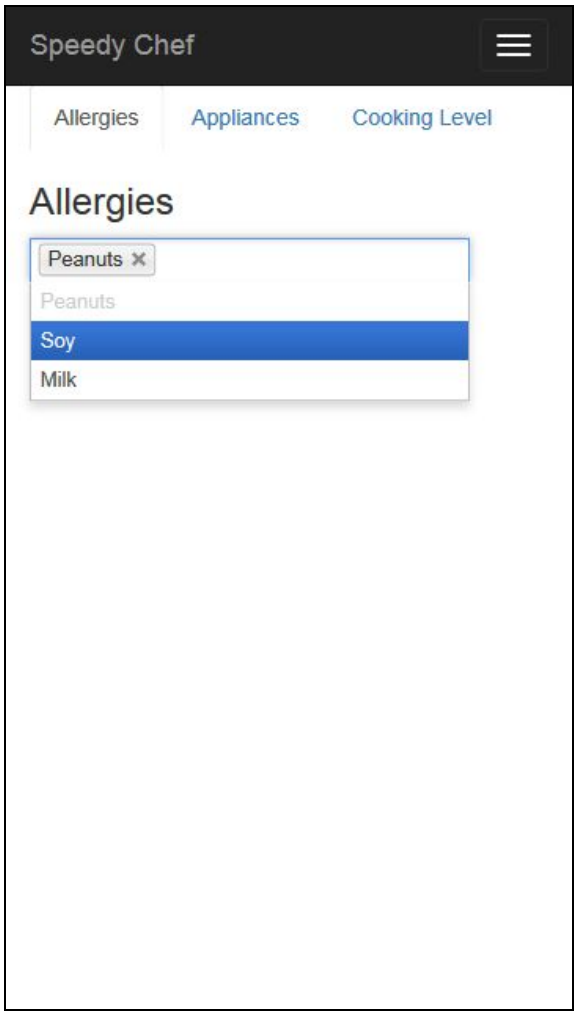


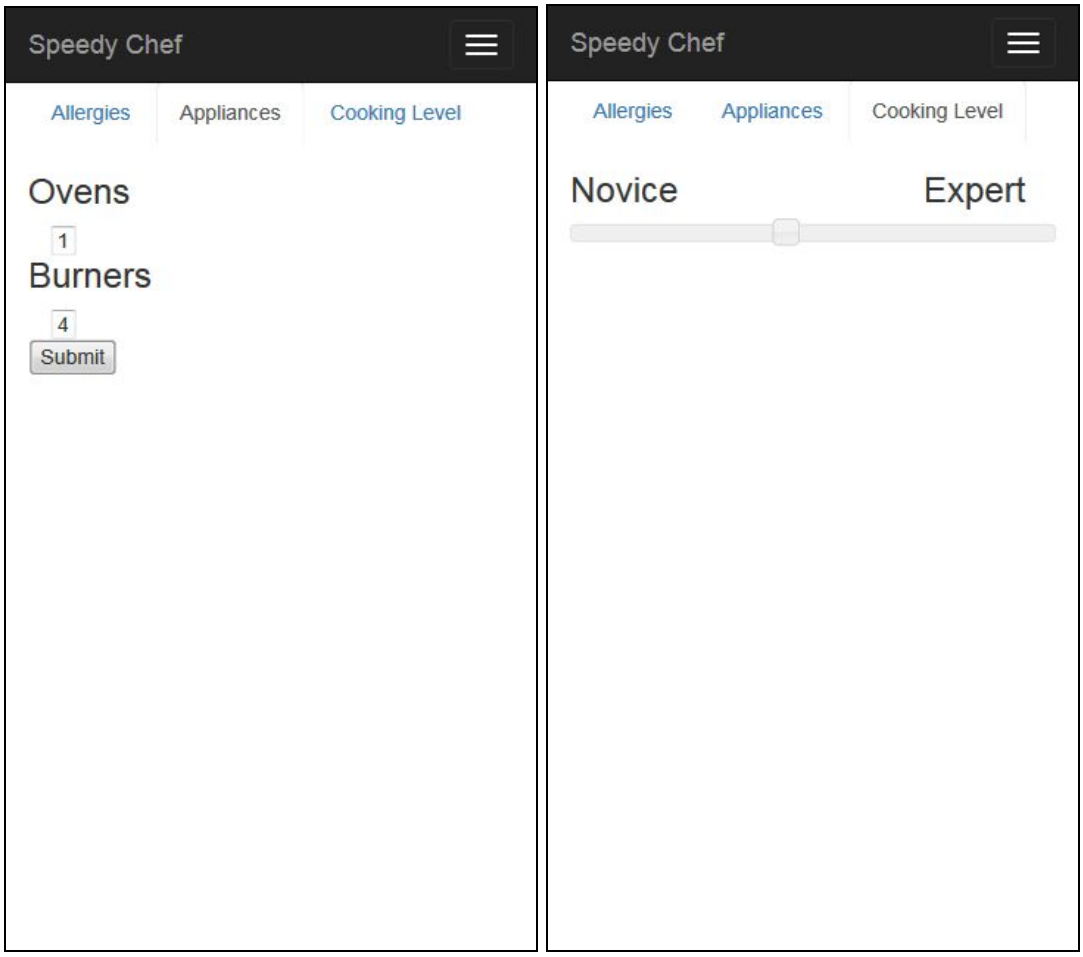

Peel Potatoes
Remove the potatoes from the boiling water. Wait for them to cool, or you can run them under cold water. Once they are cool enough to handle, use a peeler to remove the skin.
5m

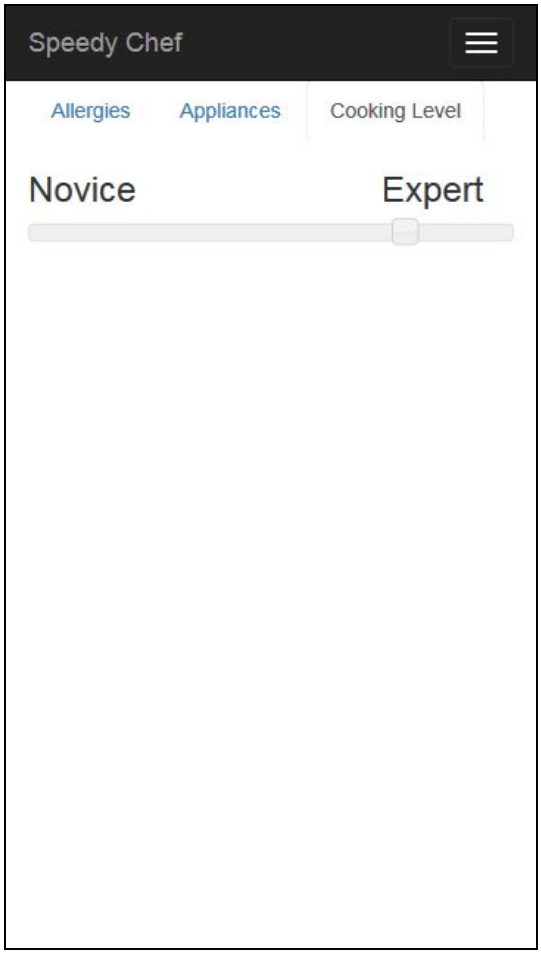
Search (work in progress)



Preferences




Speedy Chef
Allergies
Appliances
Cooking Level
Ovens
1
Burners
4
Submit
Speedy Chef
Allergies
Appliances
Cooking Level
Novice
Expert


Speedy Chef
Allergies
Appliances
Cooking Level
Novice
Expert

Coding Progress (Week 7- 9)

The major goals that we accomplished during week 7 were primarily UI-related. The design from our prototype has been translated into an Android application with very few changes. Several additional front-end features were added to make this interface perform optimally.

Using a multi-threaded Singleton class called CachedData, we can store data and request data in order to remember options and selections between Activities. This is particularly helpful in sub-Activities that directly correspond to the previous, but need data that is not of a primitive type. It is also a great way to extract out the pre-processing or data processing overhead that would bloat the more concise, direct code of the interface itself.

Using Xamarin guides, we have adapted the Android class RecyclerView to suit our needs. It allows for dynamic scrolling through a seemingly limitless set of data using a clever method of CardView generation and discarding. To tailor this View, we create a few supporting classes: a base object (such as our SideBySideObject 2x2 picture scrolling), an Adapter object that mediates between the base object code and the ViewHolder (visual, xml based objects), and the ViewHolder itself (which deals with clicks, visuals, input, etc.). Using this flow of data objects, we can create multiple different types of RecyclerView to achieve much more meaty views than the typical, native ListView and we can do so in more complex C# code that allows our XML to be more concise and less reliant on hard-coded styling.

We have yet to extract it into its own class, but we have created a default core menu that can be shared between all views. It took some creative thinking, but this menu can be customized according to two different styles within our values directory. One is for the actual menu itself while the second is specifically for ActionMenu text properties.

We also wrote a timer that can be used directly through this app, preventing the user from needing to find a kitchen timer. This is accessible from the walkthrough feature.

At the beginning of this week, we also set up a web API that will eventually supply the application with all of the relevant data. Currently, this program is capable of supplying JSON-formatted mock data upon request.

During week 9, the main goal was to fix any outstanding issues with the user interface and integrate these features with the API. This week, the appearance of the app was completely redesigned so that it would look more appealing to users that we present it to. Because this is a

cooking app, we felt that it would look more appealing if we used warm colors. Therefore, we chose orange as the primary color for our UI.

A recipe view page was also added to fill a gap in the user interface. Almost all of the features of the UI have been updated to populate their respective screens with information from the API.

Continuous Integration

We have successfully set up Microsoft Azure for hosting our API, any progress that is pushed to the GitHub Repository is automatically deployed upon completion of code review. Due to having our API in the cloud, if we were to release this app on the market, we would be able to scale it to handle any level of traffic we receive. We are also hosting our database in the cloud, this allows us to manage our app through the same portal and scale the database along with the API.

Features Completed

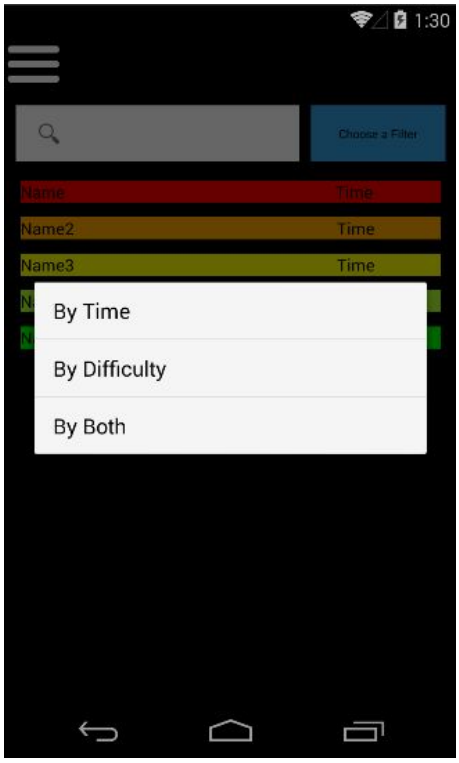
Week 8:

Search

Coder: Steve Trotta

The current status of the Search screen is that the front end is solid, with all features of the UI working as expected or better. The color scheme depending on difficulty looks fine and functions correctly while the modified *SearchView* submits string data as expected. Not *much* has changed on this front since the last sprint *visually*, but much has changed by way of code readability and preparation for integrating the API to the UI/Xamarin code. As implied, this connection has not yet occurred, but should over this coming weekend (10/30/2015). Here are some snippets of the code used for the android UI as well as an exemplary screenshot of the *Search* feature:

```
        if (tupleInQuestion.Item3 == 5) {
            vh.LeftText.SetBackgroundColor (Android.Graphics.Color.Red);
            vh.RightText.SetBackgroundColor (Android.Graphics.Color.Red);
        } else if (tupleInQuestion.Item3 == 4) {
            vh.LeftText.SetBackgroundColor (Android.Graphics.Color.Orange);
            vh.RightText.SetBackgroundColor (Android.Graphics.Color.Orange);
        } else if (tupleInQuestion.Item3 == 3) {
            vh.LeftText.SetBackgroundColor (Android.Graphics.Color.Yellow);
            vh.RightText.SetBackgroundColor (Android.Graphics.Color.Yellow);
        }
        else if (tupleInQuestion.Item3 == 2) {
            vh.LeftText.SetBackgroundColor (Android.Graphics.Color.GreenYellow);
            vh.RightText.SetBackgroundColor (Android.Graphics.Color.GreenYellow);
        } else if (tupleInQuestion.Item3 == 1) {
            vh.LeftText.SetBackgroundColor (Android.Graphics.Color.Green);
            vh.RightText.SetBackgroundColor (Android.Graphics.Color.Green);
        }
        vh.LeftText.Text = tempLeftText;
        vh.RightText.Text = tempRightText;
    }

    public override int ItemCount
    {
        get { return mRObject.NumElements ; }
    }
}

public class RecipeObject
{
    public int NumElements;
    public List<Tuple<string, string, int>> RecipeList;

    public RecipeObject ()
    {
        this.RecipeList = new List<Tuple<string, string, int>>();
        this.RecipeList.Add(new Tuple<string, string, int>("Name", "Time", 5));
        this.RecipeList.Add (new Tuple<string, string, int> ("Name2", "Time", 4));
        this.RecipeList.Add (new Tuple<string, string, int> ("Name3", "Time", 3));
        this.RecipeList.Add (new Tuple<string, string, int> ("Name3", "Time", 2));
        this.RecipeList.Add (new Tuple<string, string, int> ("Name4", "Time", 1));
        this.NumElements = this.RecipeList.Count;
    }

    public RecipeObject (List<Tuple<string, string, int>> inList)
    {
        this.RecipeList = inList;
        this.NumElements = this.RecipeList.Count;
    }

    public Tuple<string, string, int> getObjectInPosition(int position)
    {
        return this.RecipeList [position];
    }
```

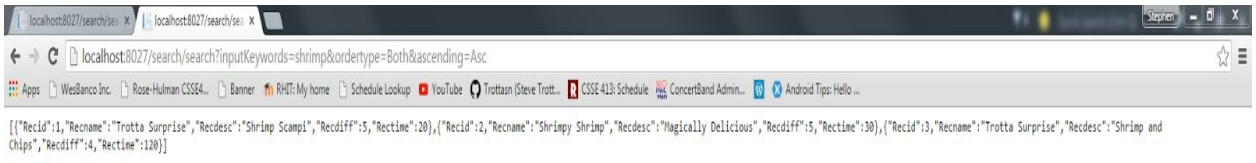


In terms of API calls and the conversion from SQL Server database query to Json code, the task is going very well. A semi-complex stored procedure called *SearchSingleKeyword* was created while simultaneously handling a slight setback caused by both an SQL Server environment and Visual Studio incompatibility with Azure or else there would have been a second procedure being made for the *HomeScreen* feature. But, this doesn't set me or the group back at all, it was simply an unexpected and unfortunate event. The API code written, specifically, takes three strings as parameters - inputKeywords, ordertype, and ascending; these parameters are all easily generated by the UI elements and will be simple to be passed. The first parameter contains all keywords to be searched for, each separated by a semicolon. This is easily split into an array of smaller strings corresponding each to a single keyword. The keywords are iterated over and each is fed, along with the ordertype and ascending parameters, into the aforementioned stored procedure. The results of this procedure are, respectively, unioned together using a custom *IEqualityComparer* called *SearchSingleComparer* in order to avoid duplicates and achieve one cohesive, ordered, table of recipes. Further filtering of unnecessary information may be done before passing back to the UI, but for now the Json generated will suffice. The code for the API and the *SearchSingleKeyword* stored procedure follow:

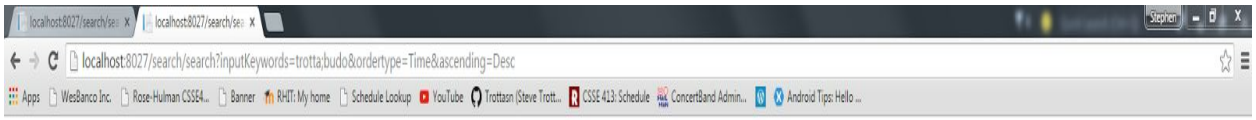
API Code for the keyword iteration [Above]

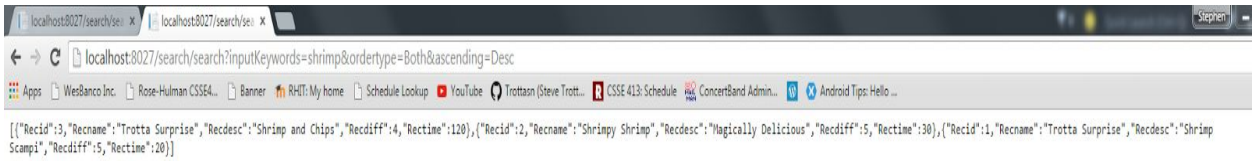
```
    public ActionResult Search(string inputKeywords, string ordertype, string ascending)
    {
        if (inputKeywords == null)
        {
            return Json(null, JsonRequestBehavior.AllowGet);
        }
        string[] keywordList = inputKeywords.Split(';');
        SpeedyChefDataContext scdc = new SpeedyChefDataContext();
        IEnumerable<SearchSingleKeywordResult> tempRes = null;
        if (keywordList == null)
        {
            return Json(null, JsonRequestBehavior.AllowGet);
        }
        foreach (string keyword in keywordList)
        {
            if (tempRes != null)
            {
                tempRes = tempRes.Union(scdc.SearchSingleKeyword(keyword, ordertype, ascending), new SearchSingleComparer());
            }
            else
            {
                IEnumerable<SearchSingleKeywordResult> firstRes = new List<SearchSingleKeywordResult>();
                tempRes = firstRes.Union(scdc.SearchSingleKeyword(keyword, ordertype, ascending), new SearchSingleComparer());
            }
        }
        return Json(tempRes, JsonRequestBehavior.AllowGet);
    }
}
```

Json code generated by an ascending ordering of *Both* the time and difficulty attributes of the Recipes [Above]



Json code generated by descending of the same attributes [Above]





Json result for trotta;budo keywords, orderingtype of only the time attribute, ascending [Above]

```
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE @restable TABLE
    (
        Recid int,
        Recname varchar(50),
        Recdesc varchar(255),
        Recdiff int,
        Rectime int
    )

    -- Insert statements for procedure here
    INSERT INTO @restable
    SELECT *
    FROM Recipe
    WHERE Recname LIKE ('%'+ @keyword + '%') OR Recdesc LIKE ('%' + @keyword + '%')
    IF @ordertype = 'Time'
        IF @ascending = 'Asc'
            SELECT *
            FROM @restable
            ORDER BY Rectime ASC
        ELSE
            SELECT *
            FROM @restable
            ORDER BY Rectime DESC
    IF @ordertype = 'Diff'
        IF @ascending = 'Asc'
            SELECT *
            FROM @restable
            ORDER BY Recdiff ASC
        ELSE
            SELECT *
            FROM @restable
            ORDER BY Recdiff ASC
    IF @ordertype = 'Both'
        IF @ascending = 'Asc'
            SELECT *
            FROM @restable
            ORDER BY Rectime ASC, Recdiff ASC
        ELSE
            SELECT *
            FROM @restable
            ORDER BY Rectime DESC, Recdiff DESC
END
```

SearchSingleKeyword stored procedure creation [Above]

Conditions of Satisfaction for Base *Search* functionality:

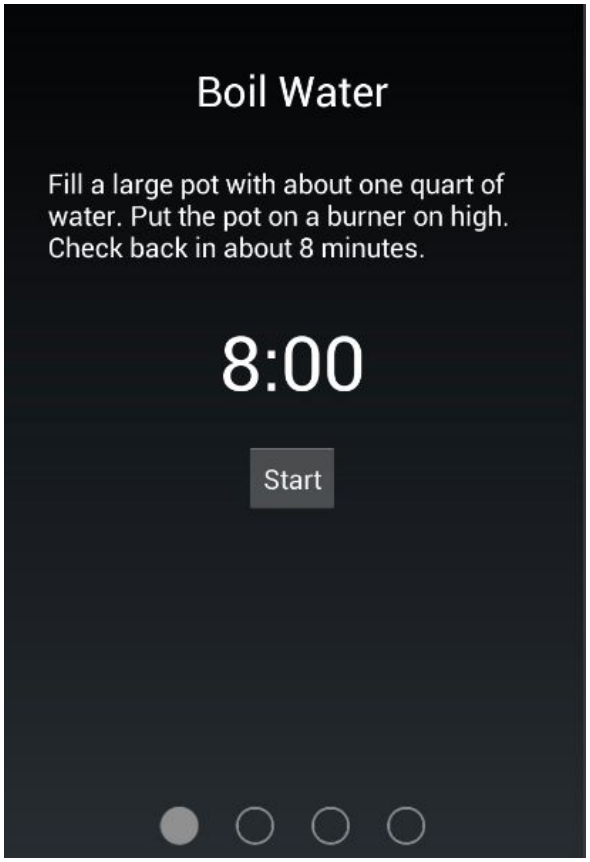
- The UI provides the ability to accept search keywords for passing through API call
- The UI provides the ability to accept order type conditions such as Time or Difficulty
- The UI displays recipes in the correct order and color codes based on difficulty
- The API calls take intended parameters and return Json code representing ordered query results
- The API theta joins single search results into one, cohesive table
- The UI layout and API methods support similar functionality for the Browse subtype of search

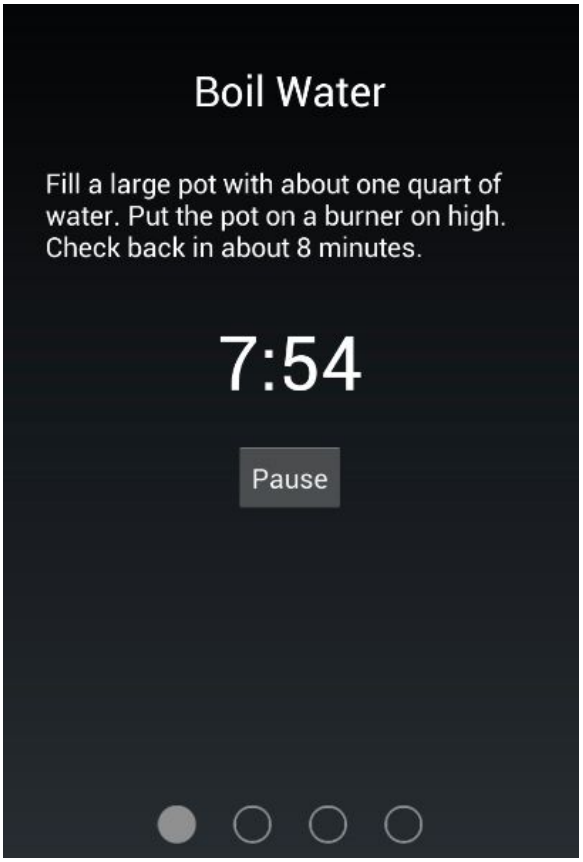
- *Unmet* is the integration of the API with the UI and the Ascending versus Descending order field

Meal Walkthrough

Coder: Alia Robinson

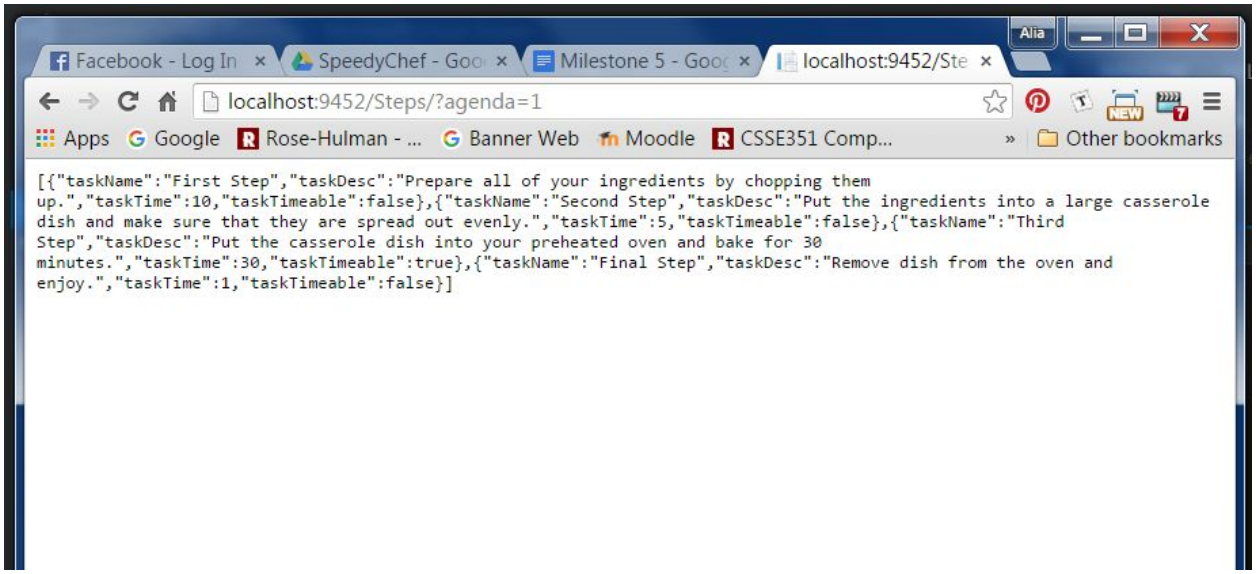
The meal walkthrough feature displays the steps necessary to complete a meal, allowing the user to swipe left and right to view the steps in the proper order. The cross-app timer feature is also a part of this screen.





This step, “Boil Water” is a timed step, so an optional timer is available for the user’s convenience.

This step, “Peel Potatoes” is not a timed step because it involves active user participation. Instead of a timer, a simple estimate is displayed.



The web API provides the client with a list of steps already in order. Upon request, it outputs a JSON-formatted array of steps. Each step has a name, a description, a time estimate, and a flag indicating whether it is timeable.

Conditions of Satisfaction:

- The UI displays the steps in a clear, readable, and ordered format
- The screen provides the user with an indication of how far along in the meal preparation plan they have progressed. This is accomplished through the progress dots at the bottom of the screen.
- The feature provides the user with a time estimate of how long each step should take

Additional Conditions:

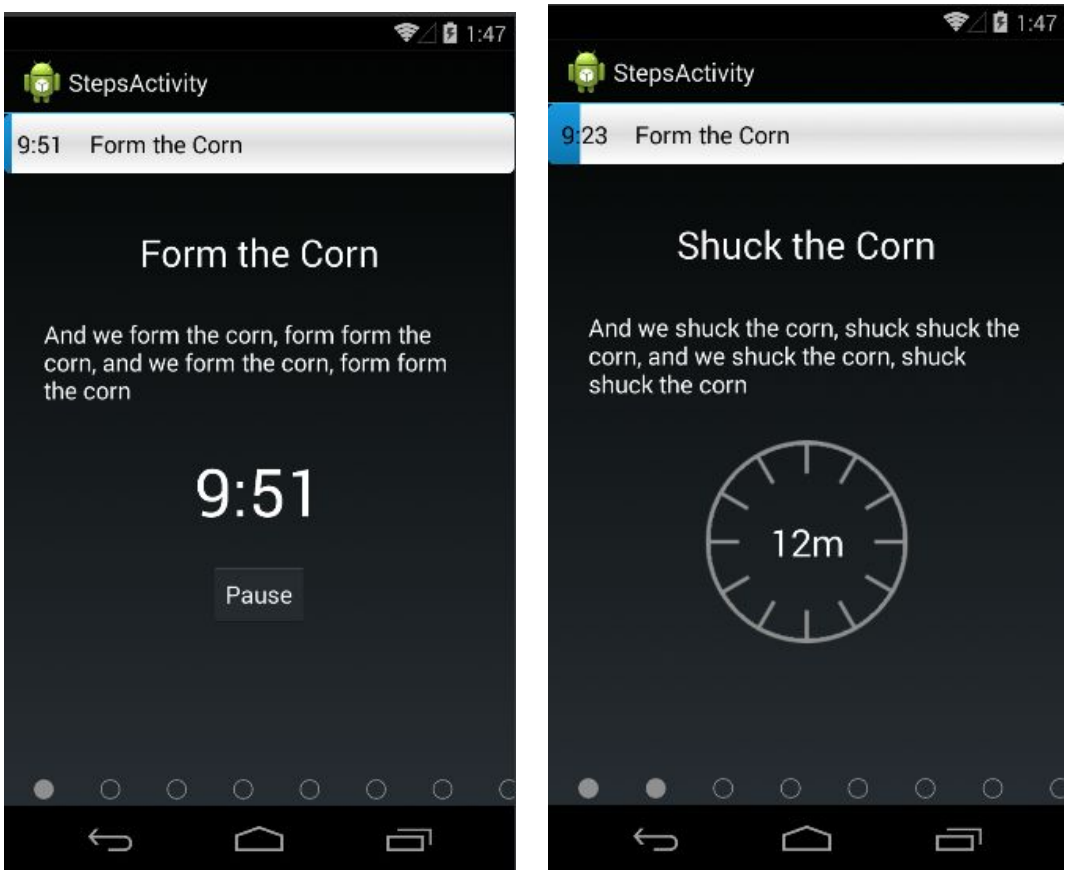
- The steps are ordered such that the recipe can be completed in as little time as possible
- All dishes in this meal are completed at the same time

The logic to satisfy these conditions will be performed by the walkthrough feature of the web API. This will be completed in a future milestone, and the UI will be integrated with this web service.

Cross-App Timer

Coder: Alia Robinson

The timer is a part of the meal walkthrough feature. As the user progresses through the steps of a meal, they may become busy with a task while waiting for another task to complete, such as manually preparing some ingredients while something else is grilling on the stove. The timer reminds them when it is time to check on their other dishes.



In this example, starting the timer for the first step brings a progress bar to the top of the screen. As the user progresses in the recipe, they can still see the remaining time for this step.

Conditions of Satisfaction:

- The timer interface is easy to read and unobtrusive
- The timer can be accessed while elsewhere on the device.
- Visual indication of how much time is remaining.

Additional Conditions:

- The timer notifies the user even when the app is inactive.

The notifications will be implemented in a future milestone. Because this is dependent on the device's operating system, we are waiting until more progress has been made on the product before implementing this final condition.

Calendar

Coder: Larry Gates
Debugger Helper: Steve Trotta

The current status of the development is in 4 parts, two front end and two back end. The two front end components are *Calendar* page and *Meal Designer* page. The two back end pieces are the *Calendar API* calls and the *Meal Designer API* calls. In the current state, all of the back end, they are in the 90% stage of development, the 10% accounting for any stored procedure or functionality that was forgotten or overlooked. For the front end, the connection to the API is needed. To dynamically load information, the connection to the API needs to happen so I can test generating the view. After the connection is set, which should be this weekend, the UI should work for the simple part. Then integration with other screens, such as the *Search* and the *Walkthrough*.

Stored Procedures:

```
-- ============================================
-- Author:		Larry Gates
-- Create date: 10/29/2015
-- Description:	Finds a given recipe name, which is from the
--				recipe table, and a given meal id number,
--				which is then used to create a connection
--				in the Meal_Recipes table, if not there.
--				Can only be called in the API AFTER
--				MealNameExists stored procedure is called
-- ============================================
ALTER Procedure [db_owner].[AddRecipeToMeal]
	@mealId int,
	@recipeName varchar(255)
AS
	DECLARE @recipeID int
	SET NOCOUNT ON;
	IF EXISTS (SELECT Recid
			FROM Recipe
			WHERE Recname = @recipeName)
	BEGIN
	SELECT @recipeID = (SELECT Recid
				FROM Recipe
				WHERE Recname = @recipeName)
	IF NOT EXISTS (
		SELECT Recid, Mealid
		FROM Meal_Recipes
		WHERE Mealid =@mealId and
			Recid = @recipeId)
		BEGIN
			INSERT INTO Meal_Recipes (Mealid, Recid)
			VALUES (@mealId, @recipeId)
		END
	END
```

This is the stored procedure used for adding recipes to a given meal on a day. This should only be called after MealNameExists stored procedure.

```
-- ============================================
-- Author:		Larry Gates
-- Create date: 10/29/2015
-- Description:	Finds if a given meal name exists,
--				returning the meal id no matter what.
-- ============================================
ALTER Procedure [db_owner].[MealNameExists]
	@user varchar(30),@date varchar(10),@name varchar(50),@size int,
	@returnValue int OUTPUT
AS
	SET NOCOUNT ON;
	IF EXISTS (
		SELECT TOP 1 Mealname
		FROM Meal, Agenda_Meals
		WHERE Agenda_Meals.Agowner = @user AND Meal.Mealname = @name AND Meal.Mealday = @date)
		BEGIN
			SELECT @returnValue = (
				SELECT Meal.Mealid
				FROM Meal JOIN Agenda_Meals on Meal.Mealid = Agenda_Meals.Mealid
				WHERE Agenda_Meals.Agowner = @user AND Meal.Mealname = @name AND Meal.Mealday = @date);
			UPDATE MEAL
			SET Meal.Mealsize = @size
			WHERE Mealid = @returnValue;
		END
	ELSE
		BEGIN
			INSERT INTO Meal (Mealname, Mealday, Mealsize)
			VALUES (@name, @date, @size);
			SELECT @returnValue = (SELECT Mealid FROM Meal WHERE Mealname = @name);
			INSERT INTO Agenda_Meals (Mealid, Agowner) VALUES(@returnValue, @user);
		END
	RETURN @returnValue;
```

Finds the meal id of a given meal for a date and user.

```sql
-- ============================================
-- Author:		Larry Gates
-- Create date: 10/29/2015
-- Description:	Deletes a meal and corresponding
--				connections from database for a
--				given day.
-- ============================================
ALTER Procedure [db_owner].[RemoveMeal]
	@mealName varchar(50),
	@date varchar(10),
	@user varchar(30)
AS
	DECLARE @mealId int;
	IF EXISTS (SELECT Meal.Mealid
			   From Meal Join Agenda_Meals on Meal.Mealid = Agenda_Meals.Mealid
			   WHERE Agenda_Meals.Agowner = @user and
				Meal.Mealname = @mealName
				AND Meal.Mealday = @date)
		BEGIN
			SELECT @mealId =
			(SELECT Meal.Mealid
			 From Meal Join Agenda_Meals on Meal.Mealid = Agenda_Meals.Mealid
			 WHERE Agenda_Meals.Agowner = @user and
				Meal.Mealname = @mealName
				AND Meal.Mealday = @date);
			DELETE FROM Meal_Recipes where Mealid = @mealId;
			DELETE FROM Agenda_Meals where Mealid = @mealId;
			DELETE FROM Meal where Mealid = @mealId AND Mealname = @mealName;
		END
```

Deletes a meal and corresponding connections if the user deletes it from the agenda.

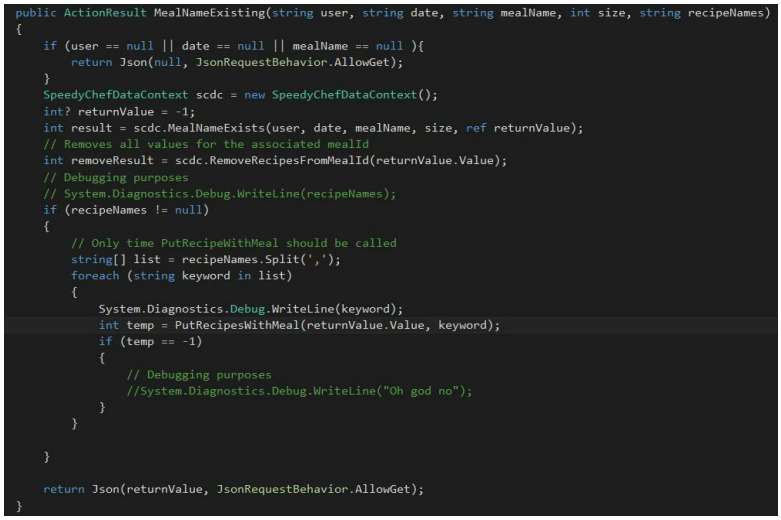
```sql
-- ============================================
-- Author:		Larry Gates
-- Create date: 10/29/2015
-- Description:	Returns the meal plan for a given day
--				to be displayed on the user interface.
-- ============================================
ALTER Procedure [db_owner].[GetMealForDay]
	@user varchar(30),
	@date varchar(10)
AS
	SET NOCOUNT ON;
	SELECT Meal.Mealname, Meal.Mealsize,
		Recipe.Recname, Recipe.Recdesc
	FROM Member mem,
		Agenda ag,
		Agenda_Meals agMe,
		Meal,
		Meal_Recipes meRe,
		Recipe
	WHERE
		mem.Memname = @user AND mem.Memname = ag.Agowner AND
		agMe.Agowner = ag.Agowner AND Meal.Mealid = agMe.Mealid AND
		meRe.Mealid = Meal.Mealid AND Recipe.Recid = meRe.Recid AND
		Meal.Mealday = @date;
```

Gets a meal for a given day for a user.

API:

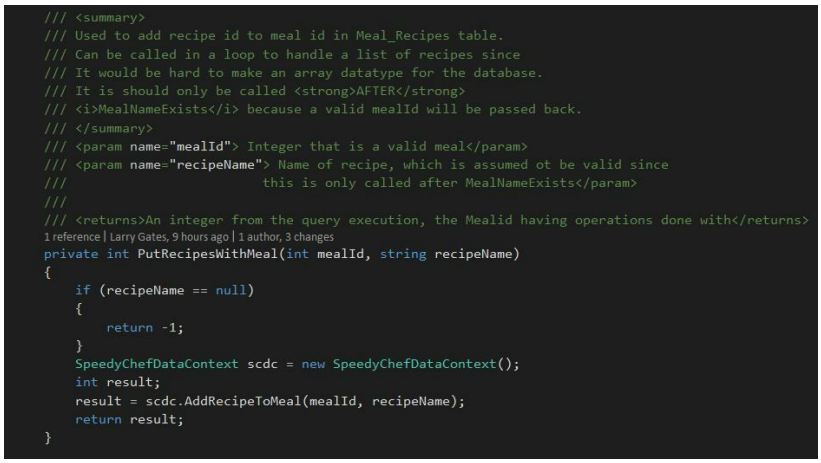
```csharp
        /// <summary>
        /// Used to call the stored procedure GetMealForDay
        /// </summary>
        /// <param name="user">Current user to get meals for</param>
        /// <param name="date">String format of day (YYYY-MM-DD)</param>
        ///
        /// <example> /CalendarScreen/GetMealDay?user=tester&date=2015-10-30 </example>
        /// <returns>JSON object that passes back information that can be used
        ///          to generate objects in the UI</returns>
        0 references | Larry Gates, 9 hours ago | 1 author, 4 changes
        public ActionResult GetMealDay(string user, string date)
        {
            if (user == null || date == null)
            {
                return Json(null, JsonRequestBehavior.AllowGet);
            }
            SpeedyChefDataContext scdc = new SpeedyChefDataContext();
            IEnumerable<GetMealForDayResult> gmfdr = null;
            // Debugging purposes
            //System.Diagnostics.Debug.WriteLine(date);

            gmfdr = scdc.GetMealForDay(user, date);
            return Json(gmfdr, JsonRequestBehavior.AllowGet);
        }
```

The API connection to the database that will execute the stored procedure GetMealForDay.



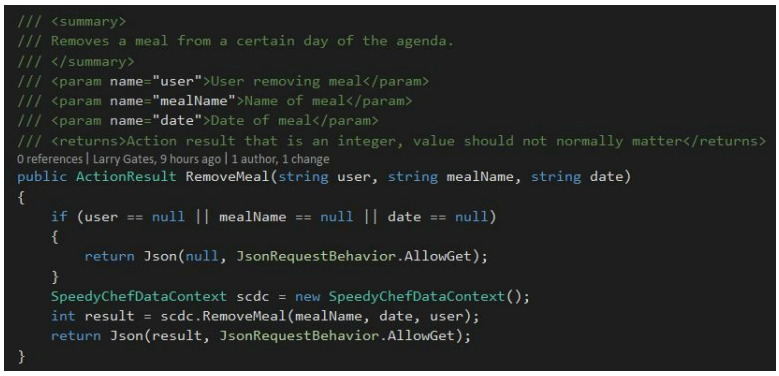
```
public ActionResult MealNameExisting(string user, string date, string mealName, int size, string recipeNames)
{
    if (user == null || date == null || mealName == null ){
        return Json(null, JsonRequestBehavior.AllowGet);
    }
    SpeedyChefDataContext scdc = new SpeedyChefDataContext();
    int? returnValue = -1;
    int result = scdc.MealNameExists(user, date, mealName, size, ref returnValue);
    // Removes all values for the associated mealId
    int removeResult = scdc.RemoveRecipesFromMealId(returnValue.Value);
    // Debugging purposes
    // System.Diagnostics.Debug.WriteLine(recipeNames);
    if (recipeNames != null)
    {
        // Only time PutRecipeWithMeal should be called
        string[] list = recipeNames.Split(',');
        foreach (string keyword in list)
        {
            System.Diagnostics.Debug.WriteLine(keyword);
            int temp = PutRecipesWithMeal(returnValue.Value, keyword);
            if (temp == -1)
            {
                // Debugging purposes
                //System.Diagnostics.Debug.WriteLine("Oh god no");
            }
        }

    }

    return Json(returnValue, JsonRequestBehavior.AllowGet);
}
```

API backend that calls the MealNameExists to get a mealId and then calls RemoveRecipesFromMealId to remove any connections with the mealId. After that, it loops through the given recipe names and adds the connection to the meal.



```
    /// <summary>
    /// Used to add recipe id to meal id in Meal_Recipes table.
    /// Can be called in a loop to handle a list of recipes since
    /// It would be hard to make an array datatype for the database.
    /// It is should only be called <strong>AFTER</strong>
    /// <i>MealNameExists</i> because a valid mealId will be passed back.
    /// </summary>
    /// <param name="mealId"> Integer that is a valid meal</param>
    /// <param name="recipeName"> Name of recipe, which is assumed ot be valid since
    ///                         this is only called after MealNameExists</param>
    ///
    /// <returns>An integer from the query execution, the Mealid having operations done with</returns>
    1 reference | Larry Gates, 9 hours ago | 1 author, 3 changes
    private int PutRecipesWithMeal(int mealId, string recipeName)
    {
        if (recipeName == null)
        {
            return -1;
        }
        SpeedyChefDataContext scdc = new SpeedyChefDataContext();
        int result;
        result = scdc.AddRecipeToMeal(mealId, recipeName);
        return result;
    }
```

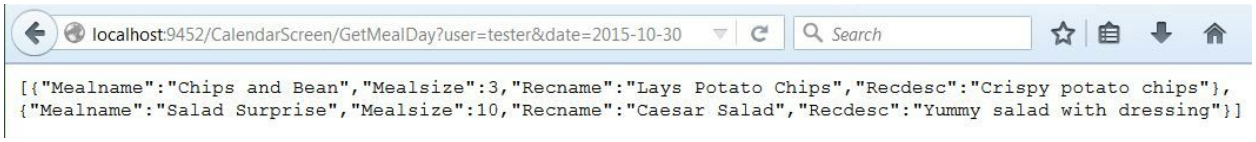
API backend call that puts the recipe with a meal.



```
/// <summary>
/// Removes a meal from a certain day of the agenda.
/// </summary>
/// <param name="user">User removing meal</param>
/// <param name="mealName">Name of meal</param>
/// <param name="date">Date of meal</param>
/// <returns>Action result that is an integer, value should not normally matter</returns>
0 references | Larry Gates, 9 hours ago | 1 author, 1 change
public ActionResult RemoveMeal(string user, string mealName, string date)
{
    if (user == null || mealName == null || date == null)
    {
        return Json(null, JsonRequestBehavior.AllowGet);
    }
    SpeedyChefDataContext scdc = new SpeedyChefDataContext();
    int result = scdc.RemoveMeal(mealName, date, user);
    return Json(result, JsonRequestBehavior.AllowGet);
}
```

API backend that will remove a meal from an user's agenda.

JSON Results:

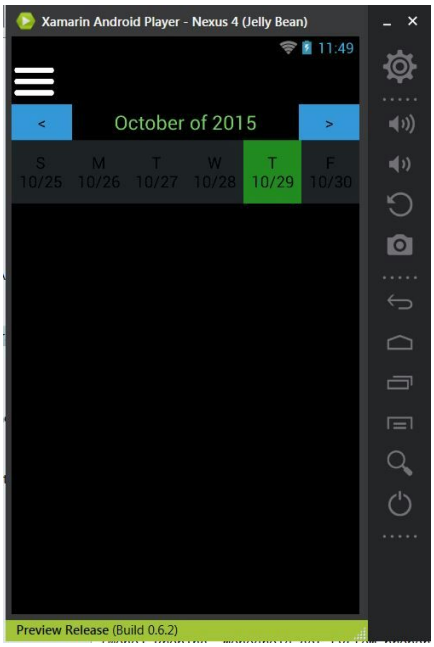


Above is an example of calling the stored procedure and getting results back.

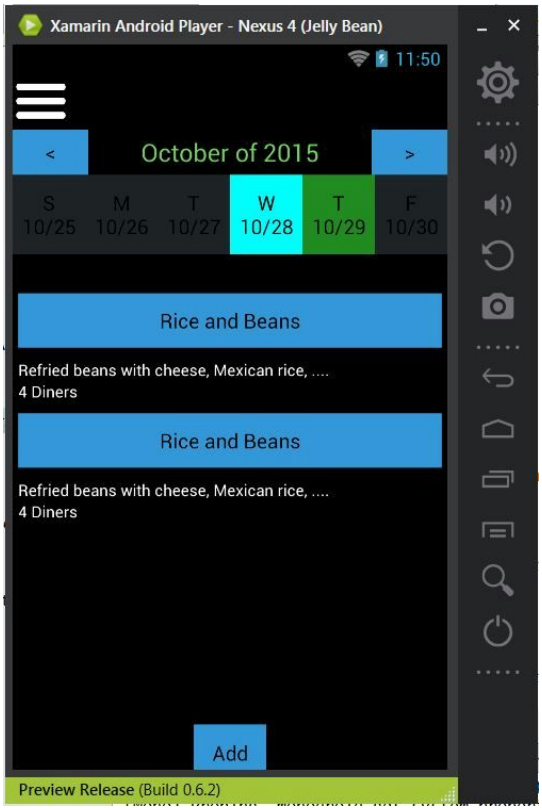
Testing:
This is an example of testing the stored procedures before implementing in with the API.

```
DECLARE @returnValue int
EXEC db_owner.MealNameExists 'tester',
⟶'2015-10-31', 'ERMAHGAD', 4, @returnValue output
SELECT @returnValue
EXEC db_owner.RemoveRecipesFromMealId @returnValue
EXEC db_owner.AddRecipeToMeal @returnValue, 'Caesar Salad'
EXEC db_owner.AddRecipeToMeal @returnValue, 'Italian Pasta'
```

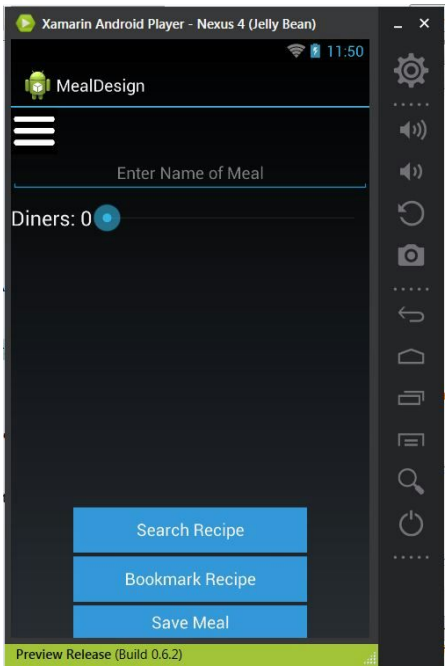
UI Screens:



The start screen of the *Calendar* page. The week navigation satisfies the ergonomic display of getting meals and is easy to navigate. Once the back end is connected to the database, everything will load once the button is clicked, handling garbage collection.



This screen, the *Calendar* screen with a selected shows mock data, which is similar to the output of JSON after a backend connection/integration is made. You will be able to click a meal and take you to the design page. Before the venture capitalist talk, the *Recipe Walkthrough* will be connected with the meal selection. The *Design* screen is also available by clicking the add button.



This screen is currently empty due to the fact of not being able to connect to the back end, which would load data in dynamically. This weekend, the *Search Recipe* will be connected to the *Search* page to allow adding recipes to a meal plan. The slider is in place as a way to maximize screen space, better than the number picker. The slider also looks more natural and appeal to a number picker. Data will be loaded if the meal is already define, allowing modifications to a prepared meal. Currently, the *Bookmark Recipe* button has no application.

Conditions of Satisfaction

Currently, the following conditions have been satisfied for the Meal Planner:

- Easy to navigate / ergonomic display for selecting this week's recipes
- Easy to read information about meal on calendar page

The following conditions will be fully satisfied after our team's code has been integrated:

- Simple way of user adding a recipe from a search to a created meal plan
- Ability to update meal plans or remove completely
- Auto erasing / garbage data collection for old days / weeks

This will be established by pulling data from the database on selected days and removing this data from the display and storage of the application.

The following has not been implemented and is not considered important for our minimum viable product:

- The ability for multiple people to enter to recipe list / shared account or planner

When social networking features are added much later on in development, this will be added.

Notes:

Additional comments about the stored procedures and the API calls can be found in the comments of the file names in the repository or on the database.

Preferences

Coder: Chris Budo

All of the stored procedures and API calls for preferences have been completed, and the UI has been created in html as a visual aid and proof of concept.

Stored Procedures:

GetGroups returns a list of all groups that the given user is a member of, requires username and password for security

```
ALTER PROCEDURE [dbo].[ADD_Allergy_User]
    -- Add the parameters for the stored procedure here
    @user nvarchar(30) = NULL,
    @pass nvarchar(255) = NULL,
    @food nvarchar(50) = NULL
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    INSERT INTO Member_Allergens(Memname, Foodname)
    VALUES(@user, @food)
END
```

```sql
-- ------------------------------------------------
ALTER PROCEDURE [dbo].[SET_StoveInfo]
    -- Add the parameters for the stored procedure here
    @user nvarchar(30) = NULL,
    @passwd nvarchar(255) = NULL,
    @tool int = 0,
    @burners int = 0,
    @pwr varchar(50)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    IF EXISTS (SELECT * FROM GET_APPLIANCES(@user, @passwd))
    BEGIN

        UPDATE Stove
        SET Pwrtype = @pwr, Burnnum = @burners
        WHERE Toolid = @tool
    END
    ELSE
    BEGIN
        INSERT INTO Stove(Pwrtype, Burnnum)
        VALUES(@pwr, @burners)
        declare @id int = (SELECT SCOPE_IDENTITY())
        INSERT INTO Group_Tool (Toolid,Groupid)
        SELECT @id, MG.Groupid
            FROM Member_Group MG
            INNER JOIN Group_Member GM
            ON MG.Groupid = GM.Groupid
            INNER JOIN Member M
            ON M.Memname = GM.Memname
            WHERE M.Memname = @user AND
                M.Mempass = @passwd
    END
```

```sql
-- ------------------------------------------------
ALTER PROCEDURE [dbo].[GET_Allergens_User]
    -- Add the parameters for the stored procedure here
    @user nvarchar(30) = NULL,
    @passwd nvarchar(255) = NULL
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT MA.Foodname, Substitution
    FROM Member_Allergens MA
    INNER JOIN Member M
    ON M.Memname = MA.Memname
    WHERE M.Memname = @user
        AND M.Mempass = @passwd

END
```

```sql
ALTER PROCEDURE [dbo].[GET_ApplianceInfo]
    -- Add the parameters for the stored procedure here
    @user nvarchar(30) = NULL,
    @passwd nvarchar(255) = NULL
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT  GT.Groupid, G.Groupname,S.Pwrtype, S.Burnnum
    FROM (SELECT MG.Groupid, Groupname, MG.Groupadmin
        FROM Member_Group MG
        INNER JOIN Group_Member GM
        ON MG.Groupid = GM.Groupid
        INNER JOIN Member M
        ON M.Memname = GM.Memname
        WHERE M.Memname = @user AND
            M.Mempass = @passwd) G
    INNER JOIN GROUP_TOOL GT
    ON GT.Groupid = G.Groupid
    INNER JOIN Stove S
    ON S.Toolid = GT.Toolid
```

```sql
ALTER PROCEDURE [dbo].[GET_Allergens]
    -- Add the parameters for the stored procedure here

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT Foodname
    FROM Food_Item

END
```

```sql
-- ================================================
ALTER PROCEDURE [dbo].[Get_Groups]
    -- Add the parameters for the stored procedure here
    @userID nvarchar(30) = NULL,
    @Password nvarchar(225) = NULL
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT MG.Groupid, Groupname, MG.Groupadmin
    FROM Member_Group MG
    INNER JOIN Group_Member GM
    ON MG.Groupid = GM.Groupid
    INNER JOIN Member M
    ON M.Memname = GM.Memname
    WHERE M.Memname = @userID AND
        M.Mempass = @Password
END
```

returns information about appliances owned by user

```
-- ---------------------------------------------
ALTER PROCEDURE [dbo].[GET_Allergens_User]
    -- Add the parameters for the stored procedure here
    @user nvarchar(30) = NULL,
    @passwd nvarchar(255) = NULL
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT MA.Foodname, Substitution
    FROM Member_Allergens MA
    INNER JOIN Member M
    ON M.Memname = MA.Memname
    WHERE M.Memname = @user
        AND M.Mempass = @passwd

END
```

Gets any allergies that the user has

Gets any possible allergies, used for the preferences page where users can specify allergies.

```
ALTER PROCEDURE [dbo].[GET_Allergens]
    -- Add the parameters for the stored procedure here

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT Foodname
    FROM Food_Item

END
```

```
-- ---------------------------------------------
ALTER PROCEDURE [dbo].[SET_StoveInfo]
    -- Add the parameters for the stored procedure here
    @user nvarchar(30) = NULL,
    @passwd nvarchar(255) = NULL,
    @tool int = 0,
    @burners int = 0,
    @pwr varchar(50)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    IF EXISTS (SELECT * FROM GET_APPLIANCES(@user, @passwd))
    BEGIN

        UPDATE Stove
        SET Pwrtype = @pwr, Burnnum = @burners
        WHERE Toolid = @tool
    END
    ELSE
    BEGIN
        INSERT INTO Stove(Pwrtype, Burnnum)
        VALUES(@pwr, @burners)
        declare @id int = (SELECT SCOPE_IDENTITY())
        INSERT INTO Group_Tool (Toolid,Groupid)
        SELECT @id, MG.Groupid
            FROM Member_Group MG
            INNER JOIN Group_Member GM
            ON MG.Groupid = GM.Groupid
            INNER JOIN Member M
            ON M.Memname = GM.Memname
            WHERE M.Memname = @user AND
                M.Mempass = @passwd
    END
```

Sets information about user's stovetop, called through preferences page when user changes information about stove type or burner number.

```
ALTER PROCEDURE [dbo].[ADD_Allergy_User]
    -- Add the parameters for the stored procedure here
    @user nvarchar(30) = NULL,
    @pass nvarchar(255) = NULL,
    @food nvarchar(50) = NULL
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    INSERT INTO Member_Allergens(Memname, Foodname)
    VALUES(@user, @food)
END
```

Adds an allergy for a user

API:

```
public JsonResult getApplianceInfo(string user, string password)
{
    SpeedyChefDataContext SCDC = new SpeedyChefDataContext();
    var json = new JsonResult();
    json.Data = SCDC.GET_ApplianceInfo(user, password).ToList();
    json.JsonRequestBehavior = JsonRequestBehavior.AllowGet;
    return json;
}
```

gets all application data from database using sproc, sends it to caller, if user is invalid, sends back empty json list

```
public JsonResult setBurnerInfo(string user, string password, string toolId, string pwrType, string number)
{
    SpeedyChefDataContext SCDC = new SpeedyChefDataContext();
    var json = new JsonResult();
    try
    {
        SCDC.SET_StoveInfo(user, password, Convert.ToInt32(toolId), Convert.ToInt32(number), pwrType);
        json.Data = true;
    }
    catch
    {
        json.Data = false;
    }
    json.JsonRequestBehavior = JsonRequestBehavior.AllowGet;
    return json;
}
```

sets information in database on user's appliance information, if it works it will return true, otherwise false

```
public JsonResult addAllergen(string user, string password, string allergen)
{
    SpeedyChefDataContext SCDC = new SpeedyChefDataContext();
    var json = new JsonResult();
    SCDC.ADD_Allergy_User(user, password, allergen);
    json.Data = true;
    json.JsonRequestBehavior = JsonRequestBehavior.AllowGet;
    return json;
}
```

adds an allergy for a user, returns true if completes

```
public JsonResult getGroups(string user, string pass)
{
    SpeedyChefDataContext SCDC = new SpeedyChefDataContext();
    var json = new JsonResult();
    json.Data = SCDC.Get_Groups(user, pass);
    json.JsonRequestBehavior = JsonRequestBehavior.AllowGet;
    return json;
}
```

Gets groups that user is a member of

```
public JsonResult getAllergens(string user, string password)
{
    SpeedyChefDataContext SCDC = new SpeedyChefDataContext();
    var allergens = SCDC.GET_Allergens_User(user, password).ToList();
    var json = new JsonResult();
    json.Data = allergens;
    json.JsonRequestBehavior = JsonRequestBehavior.AllowGet;
    return json;
}
```

gets allergies that a user have through sproc call, returns list of allergies and substitutions if there are any

```
public JsonResult getAllAllergens()
{
    SpeedyChefDataContext SCDC = new SpeedyChefDataContext();
    var json = new JsonResult();
    json.Data = SCDC.GET_Allergens().ToList();
    json.JsonRequestBehavior = JsonRequestBehavior.AllowGet;
    return json;
}
```

returns all possible allergens in json list
Json result example:


```
[
        {
                "Groupid": 1,
                "Groupname": "Lakeside 415",
                "Groupadmin": "budocf"
        },
        {
                "Groupid": 2,
                "Groupname": "Budo family",
                "Groupadmin": "budocf"
        }
]
```


enumeration of groups that user ‘budocf’ is a member of

(this is recieved by calling [base url]/preferences/getgroups?user=budocf&pass=password, once we have the api deployed, it will be accessable at speedychef.me or speedychef.azurewebsites.net)

Conditions of Satisfaction:

- Database has stored procedures that allow for all aspects of user preferences to be edited *(almost complete, some preferences potentially being stored on device)*
- The API has functions that allow for the user to add an allergy or appliance, and get information about allergies, appliances and groups *(satisfied)*
- The UI is able to make calls to the API sending and receiving data *(working in web version)*
- The UI is part of the android application
- The UI is able to make calls to the API

Week 9:

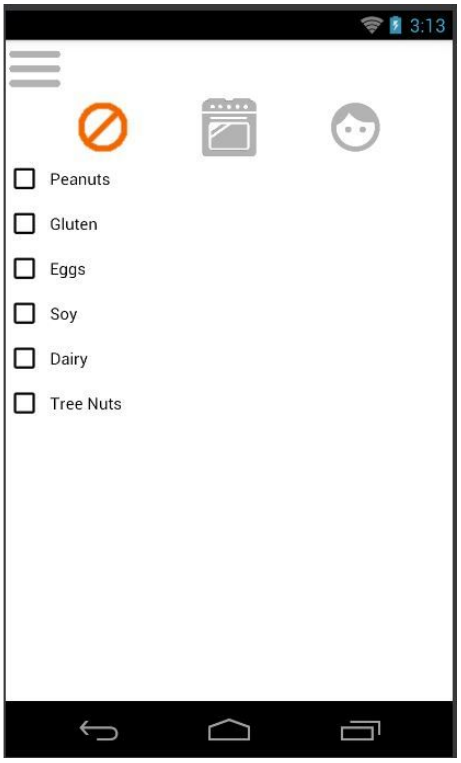
Preferences

Coder: Chris Budo

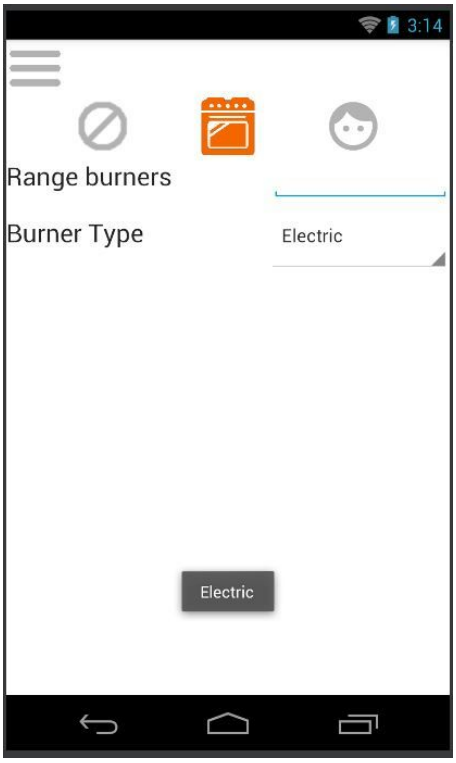
Front End Work:

preferences tabs have been built out, most have stayed close to how they were originally envisioned. The only one that has changed drastically is the allergens page, which has been changed to a scrollable page of checkboxes (this can be narrowed down through use of a search bar at a later date). The frontend has also been updated to match the new theme of the application.

Allergies page:



Appliances page:



the allergies page gathers its information from the database of allergens, searching to narrow down the list will be implemented at a later date. the appliances page will allow for the user to

input information about their stove, information that will be used to the number of concurrent steps that can be run if they require use of a burner.

Personal Information:

user is able to adjust their level of cooking skill through use of a slider.
if there are any more pieces of user information that we we gather in the future they will go here.

Backend:
The backend has been completed, and any information that is being stored in the database is able to be updated through the API. This information is going to be initially stored on the device cache, since there is currently no weight on this information in the search or walkthrough features.

Notes:
Preferences progress was slowed by resource time being re-allocated to git repository management and continuous deployment setup

Testing:
All of the API testing was done in the past milestone, this milestone, this milestone was primarily UI testing, all of which was done manually.

Conditions of Satisfaction:

- Preferences pages laid out in a way that makes sense to users
- Users able to use preferences pages without any assistance needed
- Preferences pages update database (upcoming)
- Preferences pages update from database (almost)

Search and Browse

Coder: Steve Trotta (Search/Browse), Alia Robinson (Recipe View)

Front End Work:

More additions and experimentations were made to the CachedData class in order to assist with Larry's connectivity to the *Search* feature and the passing of data back and forth between *Search, Browse, Calendar Meal Planner,* and *Design Planner*. Specifically, properties that track previous state/activity, the type of previous activities, the most recent meal and recipe id's passed, and information needed to make various decisions between activities without reliance on "EXTRA"s.

With these changes came my decision to extract shared code out into a class called CustomActivity in order to more easily adjust values within CachedData and provide a different class to inherit from besides the default Activity class. Within this top-level shared class, we can replace long sections of repeated code ("the menu click handler for example") but still have them work between activities and provide the same functionality.

Connection between *Search* and the *Recipe View* is now accomplished. The user can click a query result from search and the appropriate Recid will be passed to a handler connected to the RecyclerView. This allows for automated and dynamic adjustment to the view without sacrificing responsiveness and on-click functionality. With one stored procedure written by Alia down and one procedure to be written by myself as soon as I'm available, we will have complete frontend compatibility with viewing more detailed information about a meal and providing the option to add the appropriate recipe to a meal on the planner.

Backend:

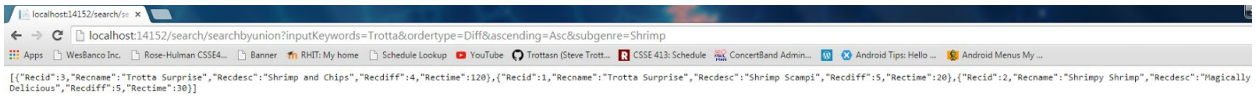
While the front-end visually may look stagnant, this is only because of extensive research and decision making on the part of the logic behind the style, including backend refinement. The API calls I had originally written were adjusted to improve performance based on recommendations consolidated from StackOverflow - ToLists (surprisingly) replaced usage of HashMaps to avoid

exceptions caused by repeated enumeration. Excepts and Intersections were used on copies of original data gathered by the increasingly more simplistic SearchSingleKeyword stored procedure. This is all in order to prevent repeated calls to the database while maintaining search complexity and helpfulness. What I mean by this is that Browse now shows all intersections between keywords and the chosen subgenre while also displaying (afterward in order) the remaining, non-duplicated subgenre results bar the keywords input. In addition, search uses regex and other forms of check and validation of input to ensure that the entire database of recipes not be sent to the front-end and overload it (potentially) with too much data. More adjustments to this end may be used in order to either get better results for the user or more likely to further boost dynamic typing performance.

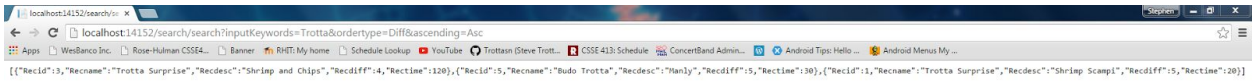
Testing:

Due to a concern about using too complex of stored procedures to query for search, I decided to make the API calls a bit more logically complex. Therefore, dummy data was added to our using various joking, but helpful variations of keywords and easy to discern descriptions.
API/webserver calls were made by browser to check for correctness before further testing was made at the UI level.

Here are two examples of the environment:



Here's an example of my testing of the Search By Union algorithm from a call to the API



Here's another example of a test call for the default (but improved) Search algorithm

You can get to this page with the proper Recid attributed and return the correct Recid to the plan page.

Conditions of Satisfaction:

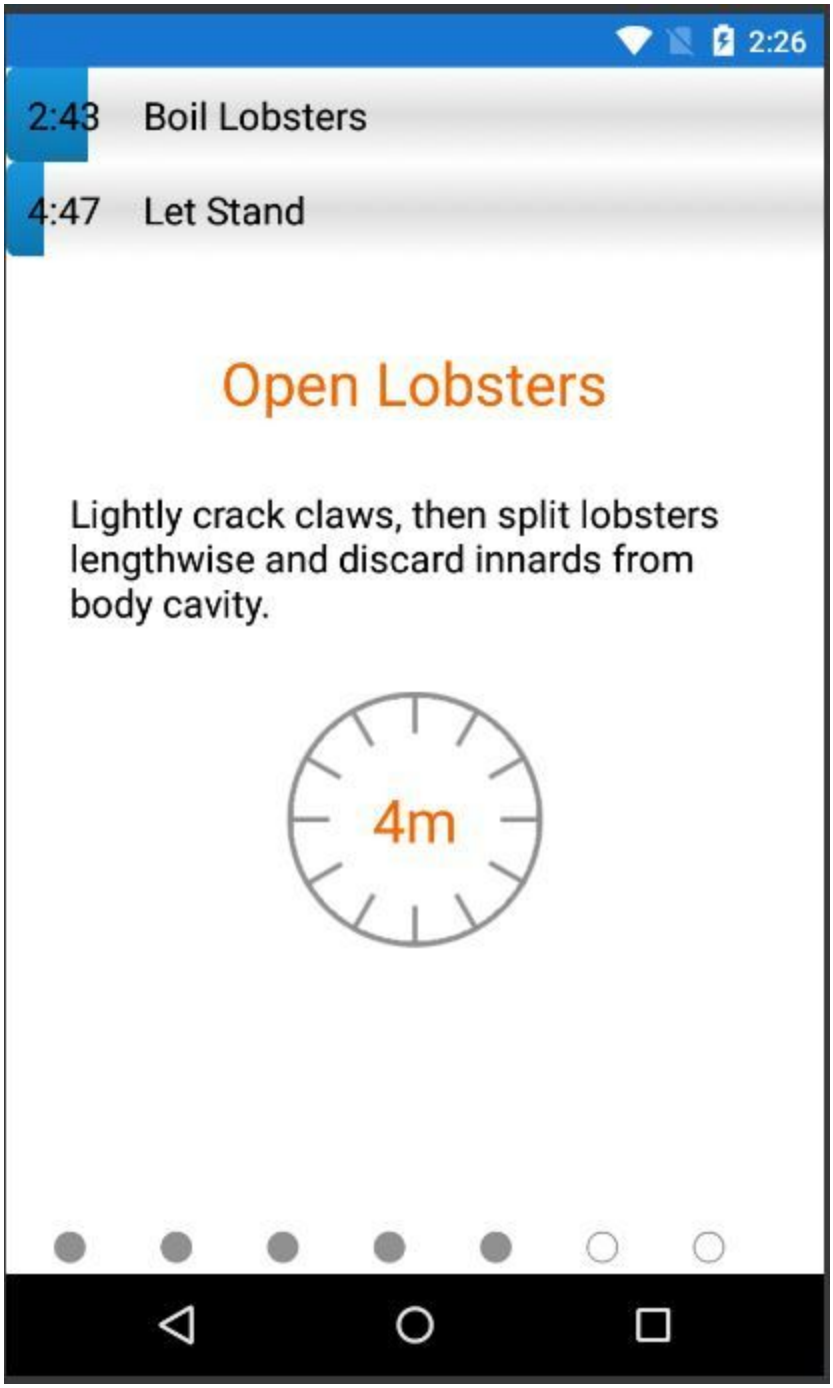
- Successful connection between the *Meal Planner* and *Search* activities
- More responsive search times and a completely tested / assured results for Browse and Search for both single and chained keywords
- Still relatively un-assured ability to pass extensive information cross-activity (we are currently working on improving cross-activity events, but they are sufficing well for now in their current state).
- With the exception of the awkward test position of search results, the user interface for Search and Browse had no issues and were well-received.

Walkthrough

Coder: Alia Robinson

Front End Work:

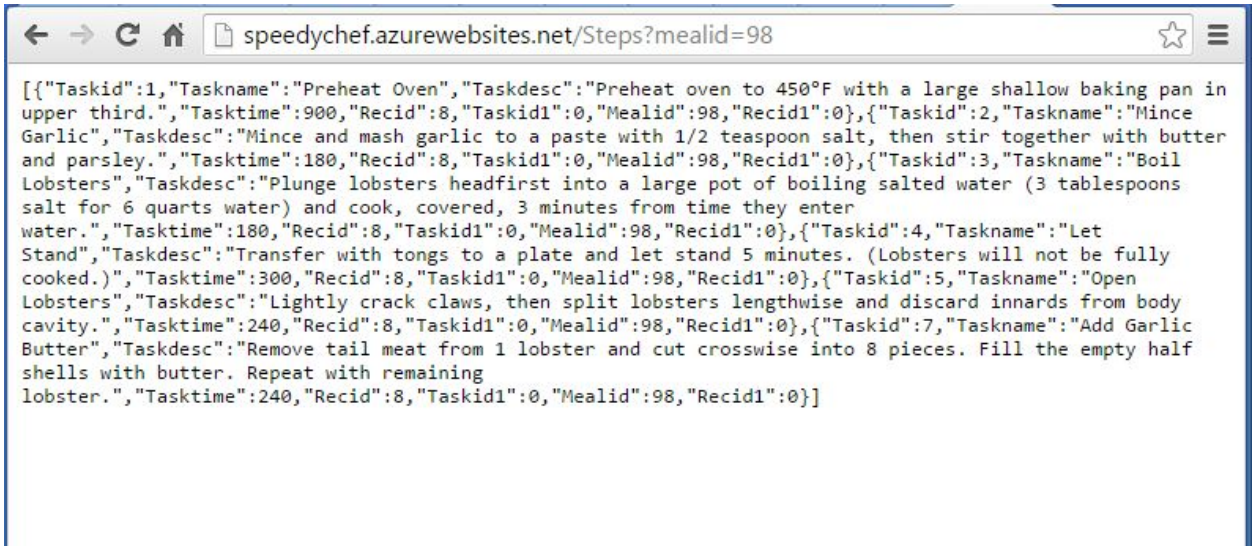
The walkthrough feature was redesigned along with the rest of the application to use lighter colors, and to look more appealing. A few bugs were discovered when adding multiple timers, and these were all resolved early on in the week. The app now supports displaying up to 5 concurrent timers. If a meal plan requires more than 5 timers to be running at once, then the timers that have the longest amount of time left will continue to run in the background, while only the 5 with the shortest amounts of time left will be visible on the screen.



An “end” page was also added to the walkthrough feature. After the user has scrolled through all the steps, a final page is displayed with the text “Enjoy your meal”, and a button that says “done”. Pressing this button closes the walkthrough. This improves the usability by making it clear to the user when it is appropriate to navigate away from the walkthrough page.

Back End Work:
The front end has been fully integrated with the API, which now collects data for the walkthrough pages based only on a meal id. The tasks are returned in the proper order for meal completion. Time optimization will be performed by the API in a future version of the product.

The following example call shows the output for a meal consisting of buttered lobster:



Testing:
The timer feature was tested on both the emulator and a mobile device. For testing purposes, a recipe was provided that used a timer for every step. Through the UI, several instances of timers were started and stopped vigorously, and all of them retained their proper values and were displayed correctly.

The API is simple to test, and this was accomplished by making calls with different meal IDs. If a meal ID does not exist, or there are no recipe tasks associated with that meal, then an empty array is returned. During our usability test, several different users created new meals that were added to the database. The API responded appropriately by providing each user with the correct data based on the meal they created in each case.

Conditions of Satisfaction:

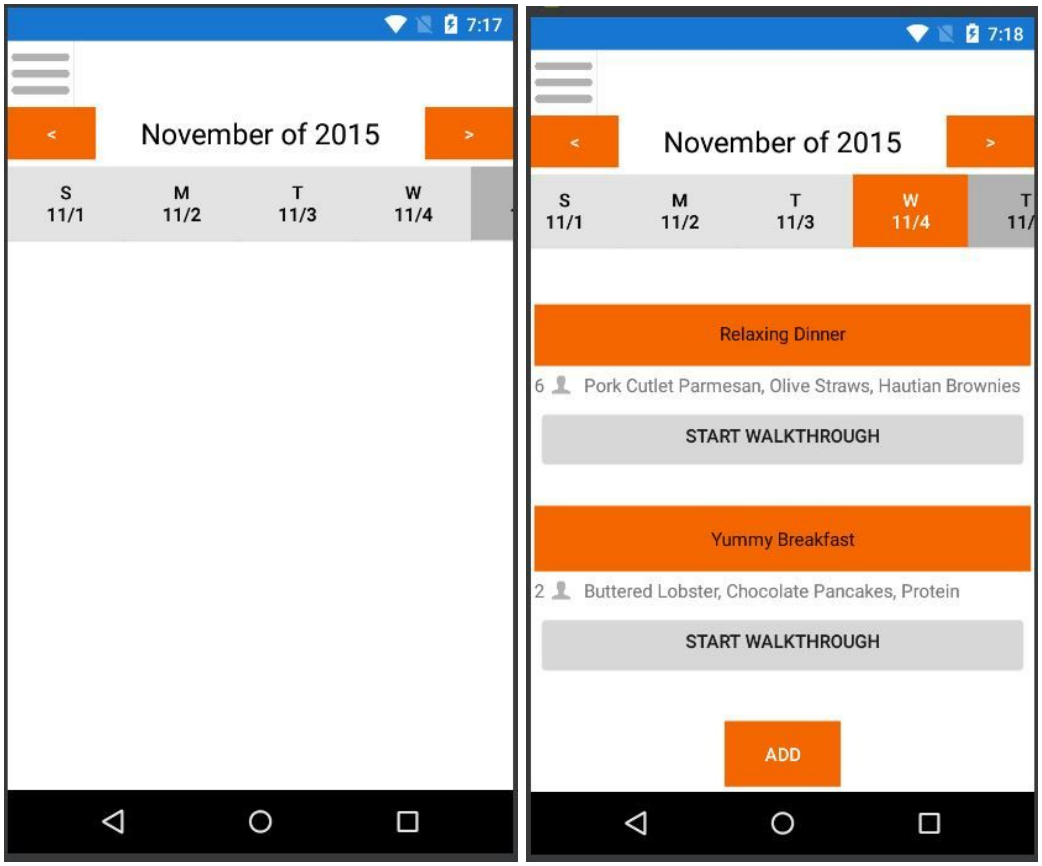
- The user is provided with a time estimate for each task and, when appropriate, a timer.
- The user can view running timers even when they are busy with another task.
- The user receives instructions that are customized to their meal plan.
- The progress dots at the bottom, the ordering of the steps, and the final page provide the user with a clear indication of how to progress through the procedure in the proper order.

Advanced Meal Planner and Designer

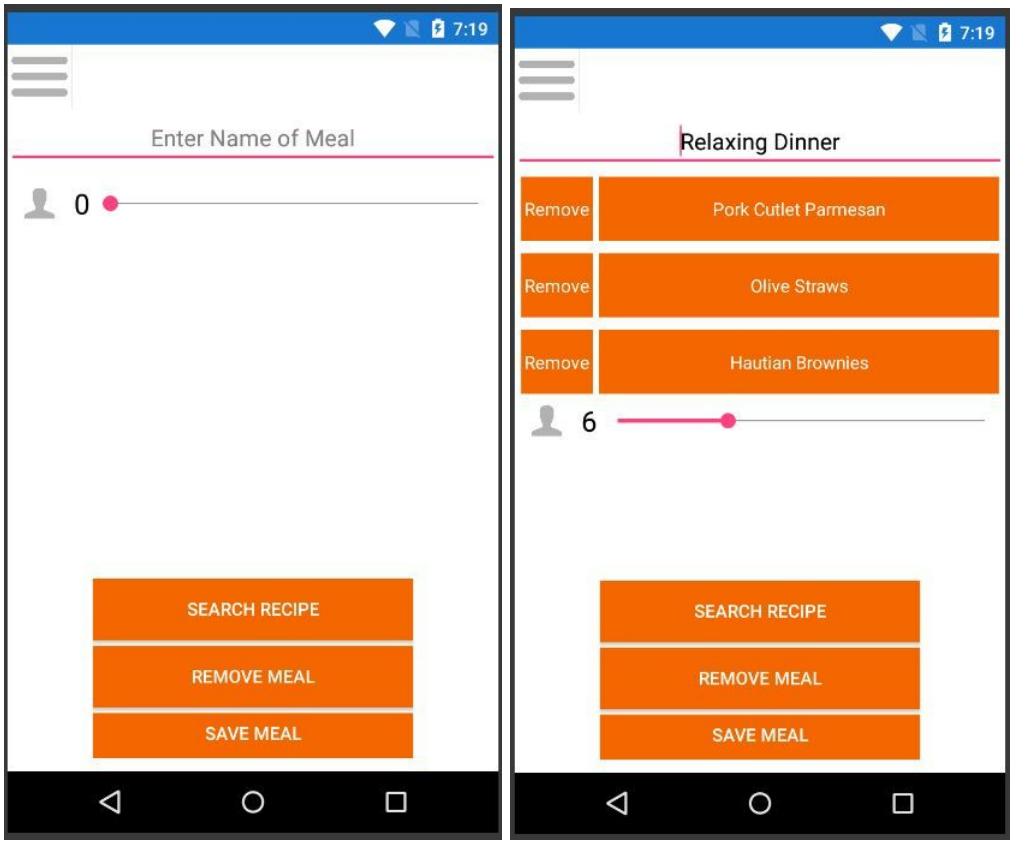
Coder: Larry Gates

(For actual code from the Android [Frontend], refer to GitHub. API Calls will have some shown)

Frontend Development:



Pictures above is the initial load screen of the *Calendar Screen* [Left screen], and the *Calendar Screen* selecting a date [Right Screen].



Pictures above is the add a new meal of the *Meal Design Page* [Left screen], and the *Meal Design Page* for an already existing meal with recipes connected to the meal [Right Screen].

Current the functionality of the pages are able to add and remove a full meal, add recipes to a meal, change the meal size, and give a meal a meal name.

Backend Development:

<u>API Procedures (Stored Procedures) Used:</u>

- GetMealDay(GetMealForDay) → *Calendar Screen*
- GetRecipesForMeal(RecipesForMeal) → *Calendar Screen*
- RemoveMealFromTables(RemoveMealFromTables) → *Meal Designer*
- AddMeal(AddMeal) → *Meal Designer*
- InsertRecForMeal(InsertRecipeForMeal) → *Meal Designer*

Refer to **Testing** to see examples of calls.

Notes:

On the actual deployment of devices, there has been some formatting issues. An example is the the Month Bar and the bar with all the days has have partly covered each other. None of the emulators have done this and I (Larry) have no physical Android device to use to constantly push and debug.

The code is commented to a great amount of detail, some areas a little empty since they were completed before the milestone and there wasn't much to finish commenting.

Testing:

Testing the stored procedures was first done in 2012 SQL Server. Using **EXEC** statements to make sure that everything was passing correctly. Once that was complete, the stored procedures were connected with the API. The API functions were tested locally then pushed to the server (supporting Continuous Deployment). Below will be various testing methods and results showing that the API calls were tested. By hand testing was a lot simpler, due to the fact that values were always changing. For more information about the API calls and stored procedures, please refer to the GitHub. All the methods in *CalendarScreenController* are documented with minimal information.

Stored Procedure Calls:

- EXEC GetMealForDay 'tester', '2015-10-30'
- EXEC AddMeal 'tester', 'Cort Pugh's Dinner', '2015-12-03', 4
- EXEC GetMealForDay 'tester', '2015-12-03' (Returns integer)
- EXEC RecipesForMeal 'tester', 29 (Returns integer)
- EXEC RemoveMealFromTables 'tester', 29 (Returns integer)

Examples for API calls:

- /CalendarScreen/GetMealDay?user=tester&date=2015-10-30
- /CalendarScreen/GetRecipesForMeal?user=tester&mealId=14
- /CalendarScreen/RemoveMealFromTables?user=tester&mealid=100
- /CalendarScreen/AddMeal?user=tester&mealname=hello&date=2015-11-05&size=5
- /CalendarScreen/InsertRecForMeal?mealId=13&recId=2

Some results for API calls:

speedychef.azurewebsites.net/CalendarScreen/GetMealDay?user=tester&date=2015-10-31


```
{
        "Mealid": 5,
        "Mealname": "Mystery Meatloaf",
        "Mealsize": 4
},
{
        "Mealid": 10,
        "Mealname": "By George",
        "Mealsize": 4
},
{
        "Mealid": 12,
        "Mealname": "ERMAHGAD",
        "Mealsize": 4
}
```


speedychef.azurewebsites.net/CalendarScreen/GetRecipesForMeal?user=tester&mealId=14


```
{
        "Recname": "Pork Cutlet Parmesan",
        "Recid": 10,
        "Recdesc": "Tender, tasty, and easy to make"
},
{
        "Recname": "Olive Straws",
        "Recid": 11,
        "Recdesc": "Bread wrapping olives"
},
{
        "Recname": "Hautian Brownies",
        "Recid": 12,
        "Recdesc": "Chocolate Brownies for recipe"
}
```


Conditions of Satisfaction:

As from the last milestone document, the following have been satisfied:

- Easy to navigate / ergonomic display for selecting this week's recipes
- Easy to read information about meal on calendar page

As of completing this milestone, the following have been satisfied:

- Simple way of user adding a recipe from a search to a created meal plan
- Ability to update meal plans or remove completely
- Auto erasing / garbage data collection for old days / weeks

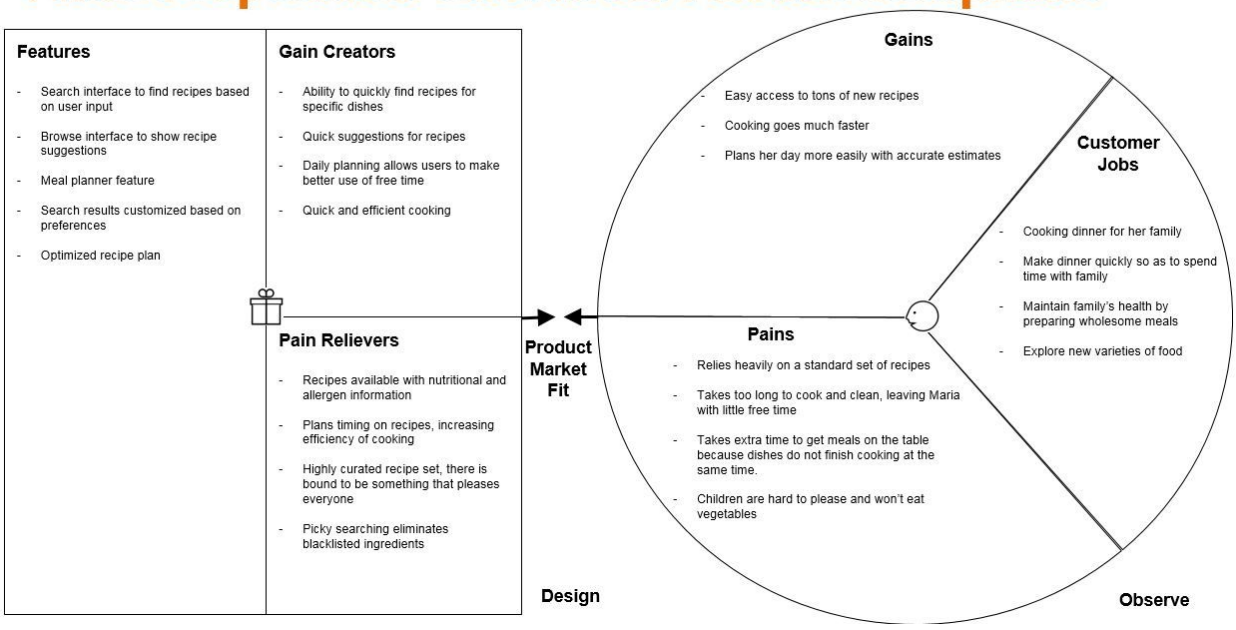
The following has not been implemented and is not considered important for our minimum viable product:

- The ability for multiple people to enter to recipe list / shared account or planner

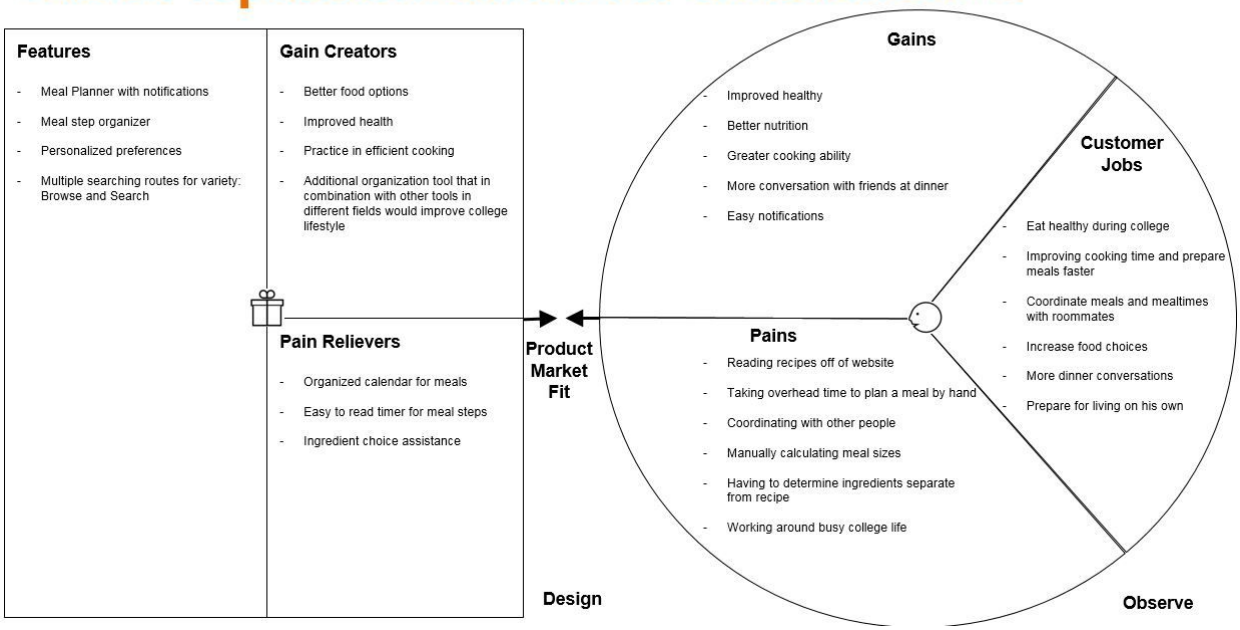
When social networking features are added much later on in development, this will be added.

Value Proposition Canvases

Value Proposition Canvas for Maria DeStephano



Value Proposition Canvas for Christian Davis



Business Model Canvas

<table>
<tr>
<td rowspan="2">Key Partners
- Local Businesses that sell potential utensils / appliances / ingredients
- Other advertisers interested in networking on SpeedyChef
- Google Play store</td>
<td>Key Activities
- Search by keywords/Database querying
- Android development and version support
- Azure based web server hosting
- C# API development in VS environment</td>
<td rowspan="2">Value Propositions
- Organized searching tool to find recipes
- Connect to meals to calendar
- Improving cooking skills</td>
<td>Customer Relationships
- App Store feedback
- Timing estimate feedback
- Accounts linked by email</td>
<td rowspan="2">Customer Segments
- Users who cook
- Recipe Providers</td>
</tr>
<tr>
<td>Key Resources
- Microsoft technology stack/suite including VS, SQL Server, Azure for VS, and Xamarin Studio for Android
- Recipe source, either professional/experienced cooks or web scraping
- Support from investors or business partners (a kickstart)</td>
<td>Channels
- Android Application
- iOS Application (Future)
- Web Application (Future) (speedychef.me)</td>
</tr>
<tr>
<td colspan="3">Cost Structure
- Database/API hosting
- Recipe testing/generation
- Continued development/support</td>
<td colspan="2">Revenue Streams
- Paid application
- Ad support
- Recipe packs</td>
</tr>
</table>

Usability Study

Goal

Our app has several features, each using its own screen. For our usability study, we wanted to determine the comfort with which potential users could navigate through the different screens and functions of our application. We focused on this navigational aspect when writing our script, and we instructed users to navigate to different pages and perform some basic tasks. While the test focused on ease of navigation, we also asked users to provide opinions on any other aspects of the application that could be improved.

The study was conducted on November 5, 2015. Larry obtained informed consent from each user and conducted the pre-lab survey. Steve then brought the users into the lab room and oriented them with the process and the android device. Steve then went into the control room and instructed the user to proceed through the list of tasks, while Chris and Alia took notes and recorded the results. Finally, Larry conducted the post-lab survey to gather any final feedback that the users had to offer.

Demographics

All of the users surveyed in this study were college students between the ages of 18 and 24. 7 of the participants were male, and 1 participant was female.

Pre-Test Survey

User	**Cooking Skill**	**Android Experience**
1	Low	Low
2	Low	Medium
3	Medium	High
4	Low	Medium
5	Medium	High
6	Medium	Low
7	Medium	Low
8	Medium	Low

Task Script

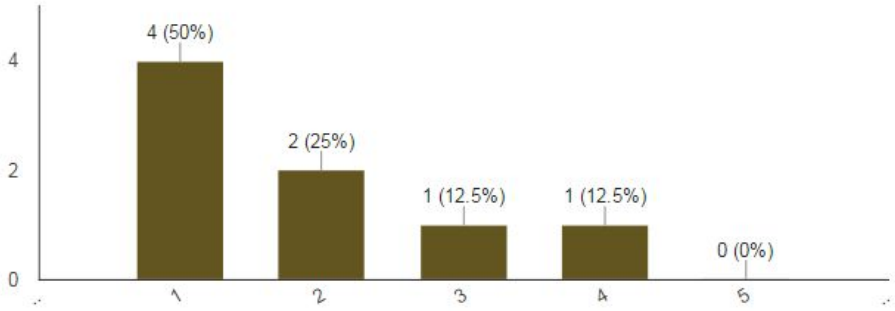
Thank you for participating in this study. You will be required to do certain tasks, and the goal of the study is to test the app, not you; so do not be afraid to make a mistake. You can take your time to do the task. If you cannot perform a task, just let me know and we will move on to the next task. Feel free to give us your feedback on your experience during or at the end of the study.

Tasks

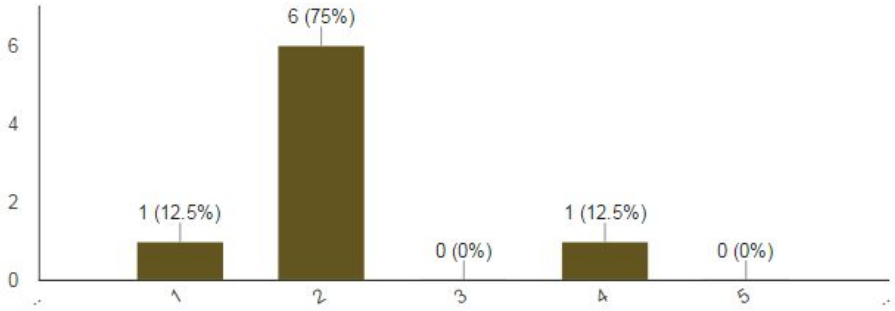
1. Go to preferences and add a peanut allergy
2. Navigate to the appliances tab under preferences and set the number of burners to 2
3. Navigate to the personal info tab under preferences and set the level of expertise with cooking
4. Use the menu to return to the home screen
5. Navigate from the Home Screen to the Plan page.
6. Find today's date
7. Click on today's date
8. Add a new meal to the date
9. Name the meal after your favorite color
10. Set the number of diners to 2
11. Click the search button, which will take you to the **Search** Screen.
12. Search for Lobster and add the recipe to the meal.
13. Save your meal.
14. Admire your new meal and return to the Home Screen
15. Navigate to the "Browse" page
16. Select Italian as nationality
17. Select Italian desserts as sub genre
18. Return to the home screen
19. Return to the meal that you created earlier with lobster
20. Begin the recipe walkthrough
21. Advance to step 3 and start the timer
22. Advance to step 4
23. Return to step 3 and pause the timer
24. Assume that you have completed all steps in the recipe. Navigate to the end of the walkthrough and click "done"

Quantitative Feedback

It was easy to navigate between pages: (8 responses)



The user interactions were intuitive: (8 responses)



Timed Tasks:

Task One:

Description:

Go to Preferences, add peanut allergy, navigate to appliances, set burners to two, navigate to personal info tab, set expertise to appropriate level.

00:00:59 seconds on average

Task Two:

Description:

Navigate to the calendar, find today's data and click on it, add a new meal, name the meal, set the number of diners to two, add a lobster recipe to the meal, and save your meal.

00:01:36 seconds on average

Task Three:

Description:

Navigate to browse and find some italian dessert recipes (alternatively some italian pasta recipes).

00:00:17 seconds on average

Task Four:

Description:

Navigate back to your created meal, begin a meal walkthrough, advance to step three, start the timer for step 3, advance to step four, return and pause the timer for step three, navigate through the rest of the recipe, and click done after completing the recipe.

00:00:58 seconds on average

Qualitative Feedback

User	Features that you like?	Features you would like changed?	Any functionality that you would add?	Would you use this product?
User 1	*The use of pictures for a lot of the browsing options makes it very appealing to use. The walkthrough pages were very neat and easy to use.*	*Fix the bugs that arose during the testing (home button not working in one of the tabs). The names of the tabs in the menu bar seem vague/repetitive. Does the cooking skill, number of diners, burners, etc actually do anything in the app?*	*The amount of features in the app is good and I wouldn't suggest adding more because that may make the app too complex for the average user.*	*If I saw someone use it successfully then yes I would use it.*
User 2	*The way it presents the instructions is really cool*	*Navigating back to a meal was difficult (it seemed like you intended to fix this in the future though anyways). I thought the meal would appear on the home screen. Also not sure if the tasks can be done concurrently or if they must be sequential? Maybe clarifying that would make it easier. (like you can view the next tasks but can't start the timer if there are dependencies until the previous timer is done.*	*N/A*	*Yeah! Super helpful that you can enter # of people, and it seems easy to follow*
User 3	*The display for the Recipes was very well done. The walk through was very intuitive and easy to follow. The font choice was spot on.*	*The Thursday header on the calendar was 'T' but this can be confused with Tuesday so it might be a good idea to change this to H or Th. While navigating through the app it did not feel as if I was moving between pages but more that every time I went to a different page, a new page was created and placed over the previous page (like a stack of cards). In browse when I selected Italian then Italian Desserts [Italian] > [Italian Desserts], but I think it might be easier or at least be more versatile in the future if you were to make the flow [Italian] > [Desserts] and just give a generic image for desserts. A way I found to do this kind of this is to actually go by tags rather than a flow (though I will say the flow felt right). In many sections of the app, buttons are stacked making it hard to select them when the font is small, it might be better if these buttons are just next to each other on the bottom of the screen.*	*Allow users to search by available ingredients*	*probably (if it adjusted meal size by total servings)*

User 4	*I liked the intuitive navigation of the application and the different features that make up the application*	*The UI needs some serious TLC, ie there needs to be many changes to the user interface*	*I don't think there needs to be any more features added.*	*After some UI changes, I would give it a try.*
User 5	*I enjoyed the timers for each step of the recipe the most. They were something that would be really useful as having more than 1 timer at once is something that's very difficult to do in most kitchens.*	*I'd like to have a swipe in from the side to have the navigation buttons pop up. The top left corner of the screen is the hardest to reach with the right thumb (which is what most people use to navigate their phones). Also, maybe the buttons being aligned in the bottom left would be better than what it is currently.*	*I'd like the ability to add timers on an empty screen to keep track of things without using an existing recipe.*	*After becoming more familiar with the application, I would definitely use it for recipes I haven't made before.*
User 6	*pretty easy to add meals and follow the steps*	*n/a*	*highlight the menu the button is easy to miss*	*depends on level of complication*
User 7	*clear menu and most pages were intuitive to use*	*make some of the buttons easier to press. particularly like adding / saving a meal*	*suggest meals based on previous choices*	*probably not*
User 8	*The add to menu feature was very nice*	*The buttons were very difficult to use, and some were not super clear in their functionality (The preferences)*	*Search recipes by price of ingredients*	*Possibly. But moot, since I do not have a smartphone.*

Findings

Finding One:

Preferences not reachable from all screens (general menu navigation concerns)

Recommendations:

Convert menu to side drawer, this will allow for menu to be accessed from more than just the button in the top, common over all pages.

Supporting Observations:

User	Scenario	Observation ID	Observation Text
User 03	Task 04	Quicklog	Navigation Issue
User 06	Task 01	11	Problem going home and getting back to preferences

Finding Two:
Search bar is not visible on search page, change to different, more contrasting color.

Recommendations:
Change the color of the search bar to match the general color scheme.

Finding Three:
The buttons on meal planning small makes it difficult for users with large, shakey, or messy hands to click buttons - also not easy to click quickly

Recommendations:
Change the button size / bigger is better

Supporting Observations:

User	Scenario	Observation ID	Observation Text
User 04	Task 04	10	There was a problem with saving the meal, steve to the rescue
User 06	Task 02	13	Save button hard to press

Finding Four:
Meal did not save / number of diners were not saved.

Recommendations:

Get stuff done, get all of the features working fully.

Supporting Observations:

User	Scenario	Observation ID	Observation Text
User 04	Task 04	9	Meal did not save
User 04	Task 04	10	Problem with saving meal, Steve to the rescue
User 06	Task 02	12	Did not save the number of people

OVO Project

The results of our usability test can be viewed at the following location:

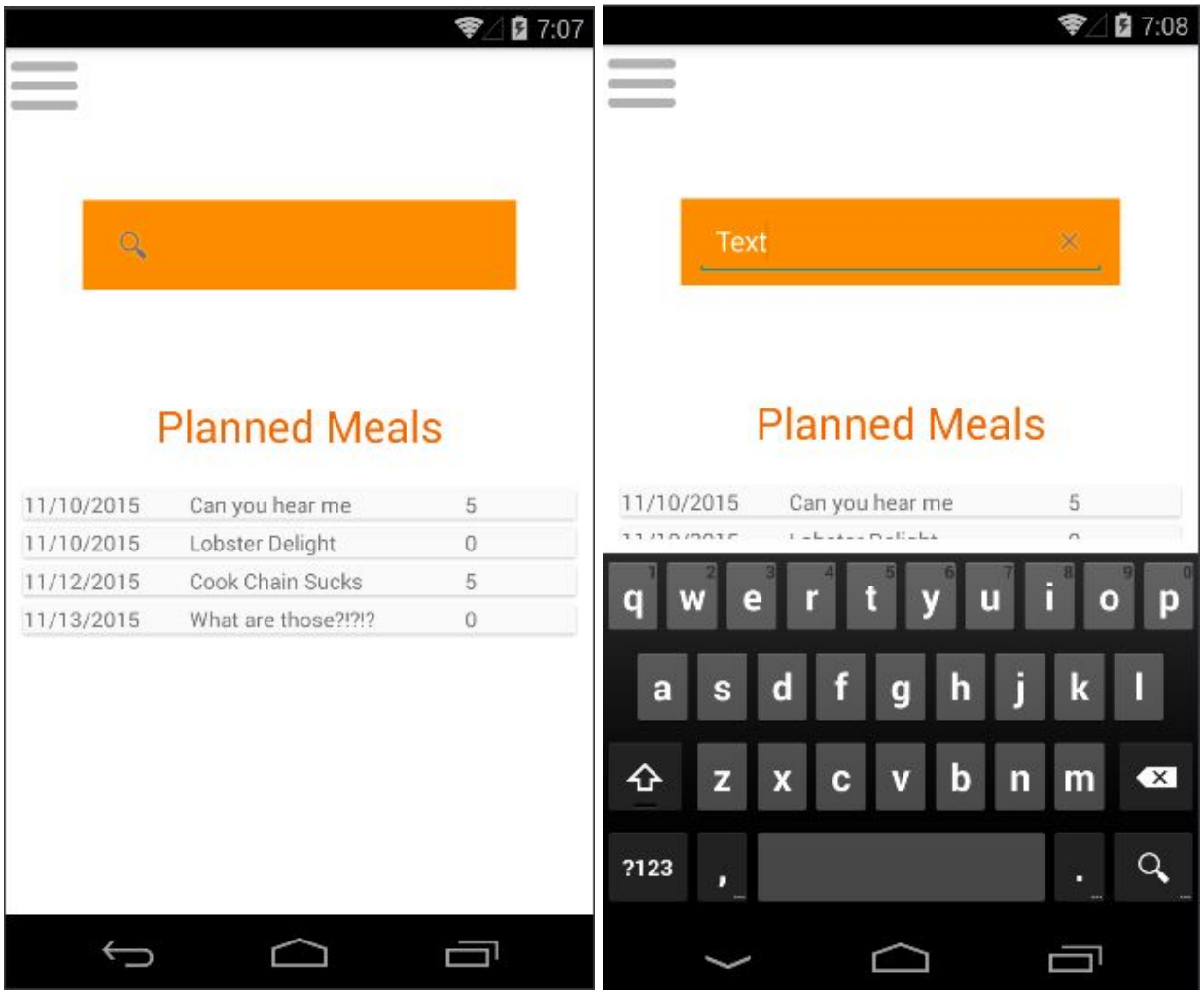
User\Usability\Documents\CSSE371\1516\Section02\TeamSpeedyChef\SpeedyChef

Usability Study Improvements

Summary

Overall, the users seemed to navigate through the application with only a few minor hangups. The biggest issues were two bugs that we discovered during the test. A few of the users had problems when trying to save a meal that they had created. There was also a case in which the user navigating away from the preferences page and returning to the home page prevented them from returning to the preferences page. Both of these bugs have been addressed. We also fixed some other display malfunctions that occurred, such as text overlapping other text or a panel not being fully visible. All additional changes were based on the feedback we received in order to best tailor the user's experience to suit their needs.

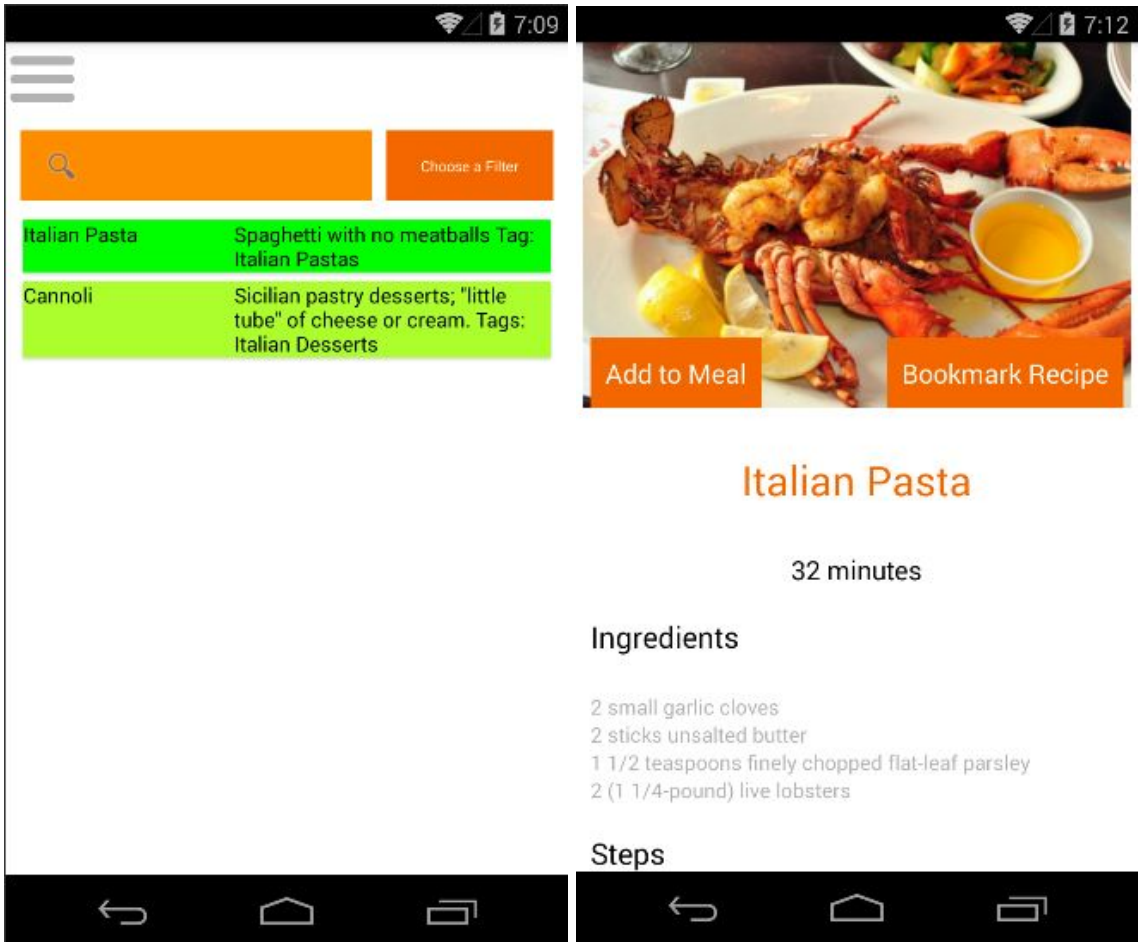
Home and Home to Search Features:



[Above Left] I decided it was time to implement the actual loading of real data / real meals for a user into the planned meals RecyclerView. That is what you're now seeing there. In addition, I changed the color scheme and general layout to make the cards look less "clickable" in an attempt to quell some of the behavior seen during the usability study. We would prefer those items stay unclicked and unclickable.

[Above Right] I also implemented the auto-navigation from the home screen to the recipe search screen on submission of SearchView input. In addition to fulfilling this feature, I changed the SearchView color to be different but thematic from the white background. (this is *actually* much harder to figure out / do correctly than would be assumed from a simple color change)

Search Feature:



[Above Left] The search bar, just like the home screen's bar was turned to the dark orange / white text theme by usability study request. The color scheme is kept for the difficulties, but the text layout was refined so that the descriptions are aligned and the titles are given more space. In addition, the ability was provided to click on objects in the search RecyclerView to generate the screen seen on [Above Right]. While the [Above Right] picture may not look like much because of the lobster for "Italian Pasta", we have queried the database for the title and therefore have the capability (easily, just with more time given to us) to generate everything else that you see.

Also with [Above Right], we have the *Add to Meal* subfeature fleshed out and working completely. By pressing that button, you are adding the recipe either to the default today's meal (from Home) or the meal chosen in the *Meal Planner* portion of the app.

Calendar Feature:

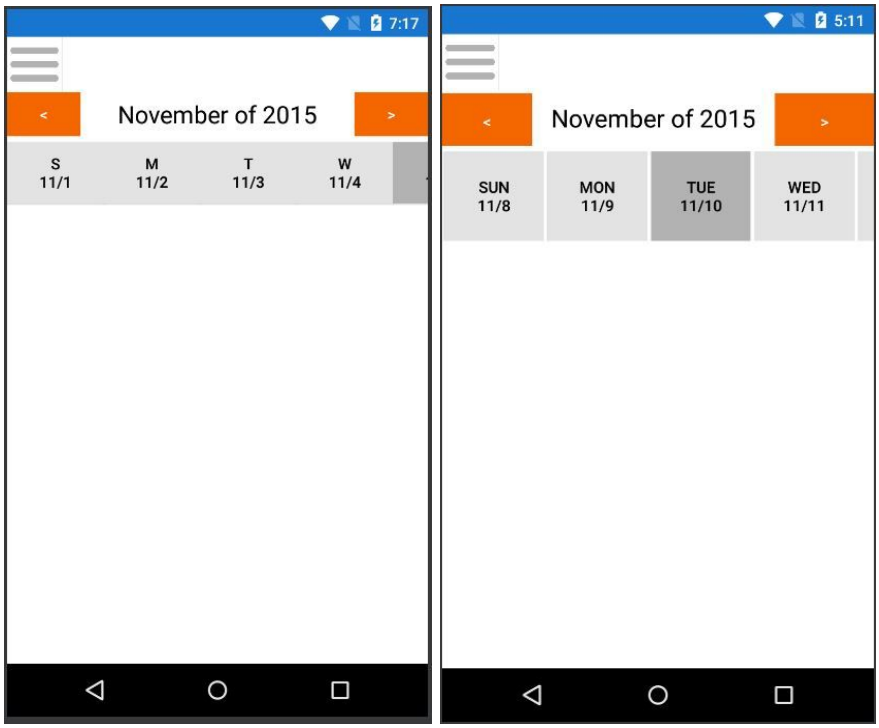
The main improvement for the *Calendar Screen* and and the *Meal Design Page* were to increase the button sizes on those pages. As the buttons on the emulator appear large enough, they were difficult for many users that did not have steady hands or smaller fingers. The buttons were increased in size to give a little more height and width. The day buttons were set to a large height but did not appear to be changed as much. The same applies for the navigating **Left** and **Right** a week, those sizes were changed but did not feel to be changed.

Another change implemented was how the days were presented. It was noted that having 2 letter **T**s could be confusing for what day of the week it was. After conversation with several people, a fair number of users prefered the single letter, and would not be confused with 2 **T**s on the weekday viewer. A fair number of users thought it would be confusing having 2 **T**s in order of a week, so it was opted to change the days of the week to 3 letters compared to 1 letter.

A significant problem that was solved was why the emulator was not displaying the *Calendar Screen* month banner being overlapped by the weekday banner but an actual device was. After messing with layout and getting a device to work, the banner overlapping was removed. Below are the changes to the following pages in the form of screenshots, the left being the **BEFORE** screenshot, where the right showing the **REVISION** screenshot. The screenshots are from an emulator.

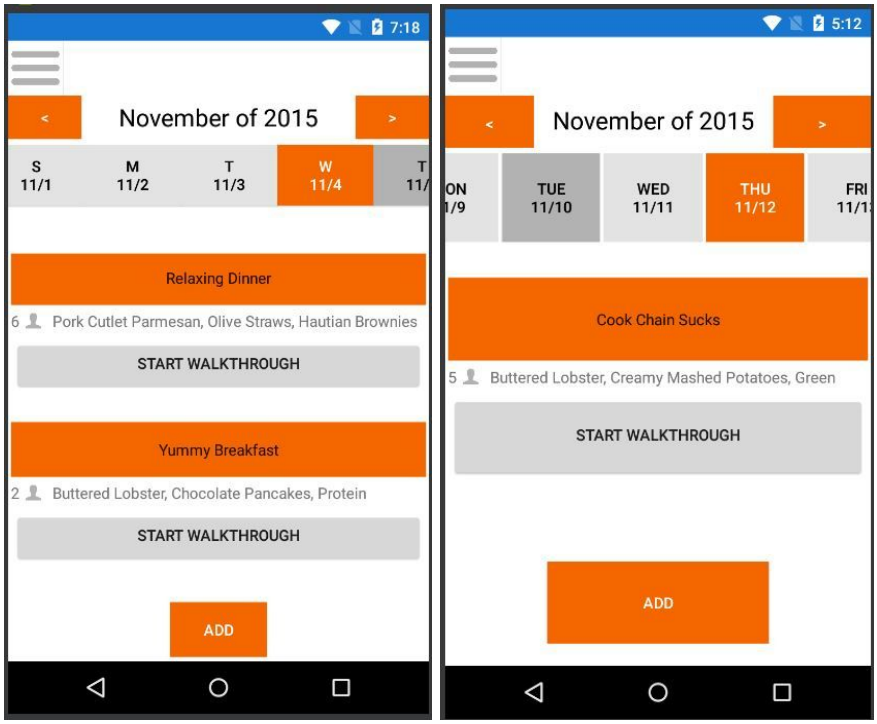
Calendar Screen without Meals

BEFORE **AFTER**



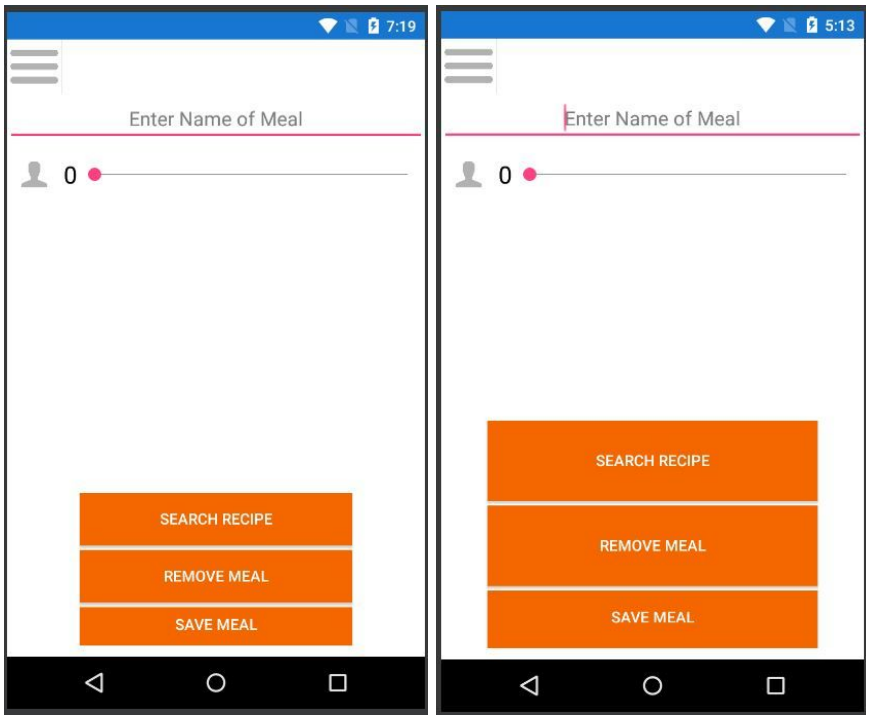
Calendar Screen with Meals

BEFORE **AFTER**



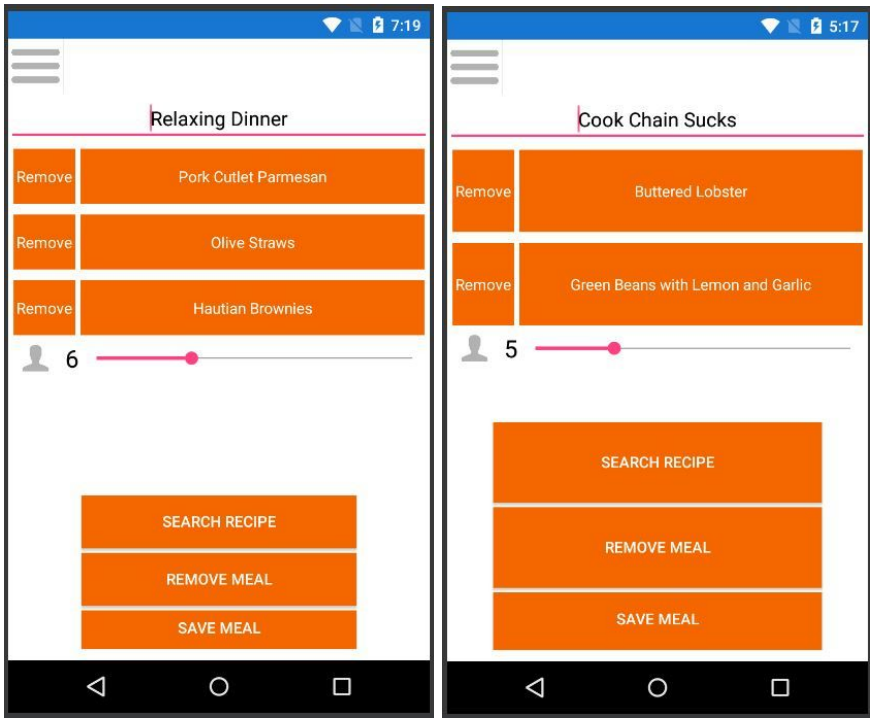
Meal Design without recipes or name

BEFORE **AFTER**



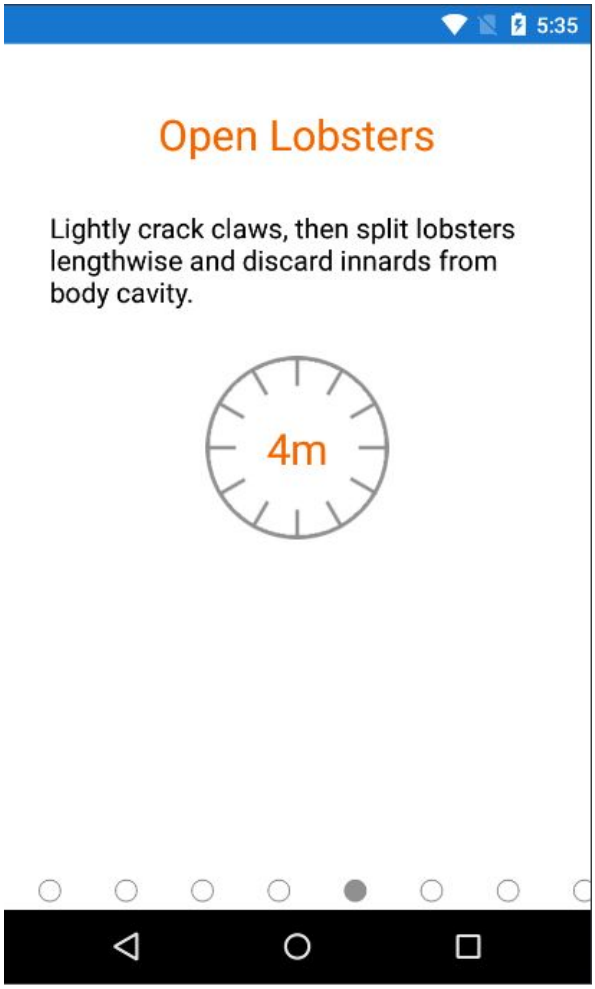
Meal Design with recipes and name

BEFORE **AFTER**



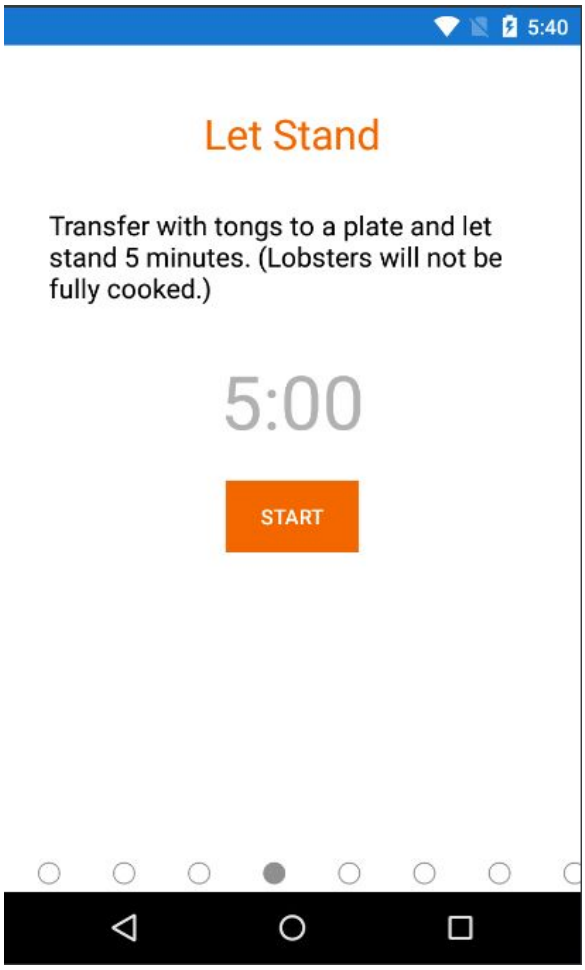
Walkthrough Feature:

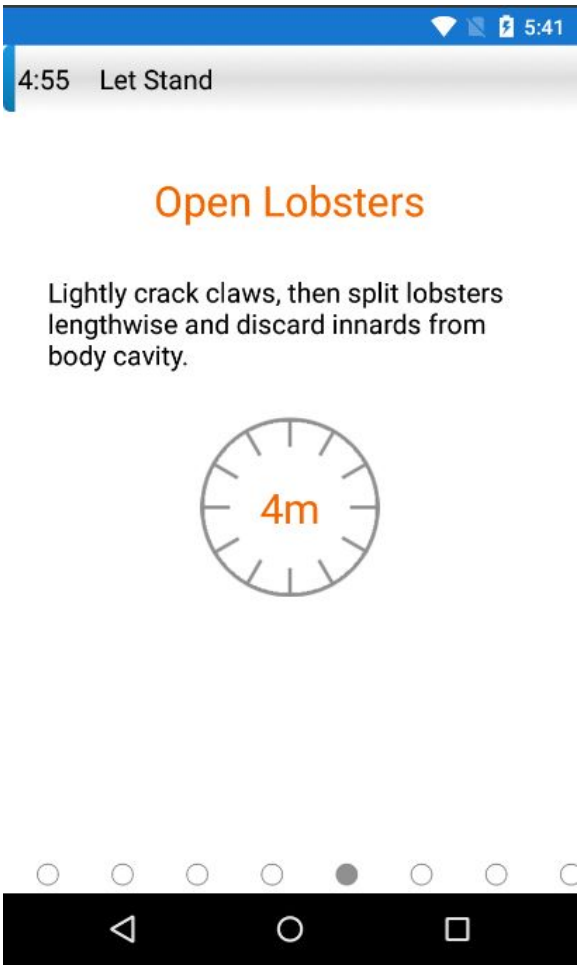
A few minor improvements were made to this feature to enhance ease of navigation and clarity of instructions. During the usability lab, a user attempted to advance from one step to the next by pressing the dots at the bottom. These dots were intended to serve as a progress indicator rather than a control. After the study, we adapted this display to allow the user to select a page by tapping a dot.



One other source of confusion became evident in the study: the clarity with which the instructions are presented. One task involved navigating to a step that included a timer, and then advancing to the next step. Some of the users paused after doing this, because their instincts told them to wait for the timer to finish counting down before progressing any further. Additionally, one user explicitly indicated in their feedback that they were not sure whether they were

supposed to wait for the timer or move on to the next step. If the user can move onto the next step while waiting for the previous step to finish, the intention of the app is for them to do so. To make this more obvious, starting a timer now automatically advances the user to the next step.





Preference Feature:

Something that was mentioned by our users throughout our usability test was that they wanted to use the back button to navigate out of the preferences windows. This was initially removed due to a problem with the tab structure in Xamarin, after the usability test, more research was done and we were able to find a way to allow users to use the back button in preferences.