

# Toque'd Cooking Assistant



Mike McDonald  
CSSE 490 Android Development Project Proposal

October 8, 2012

# 1 Project Description

Toque'd is designed to integrate step by step recipes and shopping lists into one simple, easy to use meal planning tool that allows overstressed students, parents, and the even culinarily uninclined to quickly plan, shop for, and cook delicious meals. Toque'd is broken into three distinct components, each represented by common kitchen fixtures: a fridge, which stores ingredients currently on hand and a shopping list of things necessary for planned meals; a stove, which allows users to find recipes by meal or ingredient; and a recipe book, which allows users to create, edit, and delete recipes that can then be cooked.

Toque'd simplifies the meal (or snack) planning and cooking processes by providing step by step instructions with photographs, shopping lists by meal, and an easy rating system for future reference, as well as extra features such as random meal selection from ingredients on hand (or suggest the recipes with a minimum number necessary) or the ability to share recipes with other friends who use the app or export recipes via email.

## 2 Requirements

- Hardware and Android
  - Must run Android 2.3 (Gingerbread).
  - Optimized for phone form factor (4.3" screen).
  - Store all data on internal SQLite database or file system.
  - Consider using RecipeBook XML, or other standardized recipe markup language, for storage and data transfer.
  - May be able to store app and data to SD card (especially larger and more intensive recipes).
- Refrigerator
  - Store items currently in stock on a shelf.
  - Maintain a list of items that must be bought.
  - All items must have a quantity associated.
  - Based on the recipe used (and amount of the recipe used), possibly diminish the quantities of the items on the shelf.
  - Consider allowing users to set alerts when levels of an item get low, or support an auto add to the shopping list.
- Stove
  - Sort recipes by meal type, main ingredient, dietary restriction, user rating, etc.
  - Support a “random” function that spits out a recipe at random from items in stock. If no items are in stock, pick on that requires the minimum number of items to be complete.
  - Directs user to a series of step by step instructions through a recipe (complete with pictures).
  - Possibly integrated timer support within the recipes or through a separate menu.
- Recipe Book
  - Have options to create a new recipe, edit an existing recipe, or delete an old recipe.
  - Have an option to simply view a recipe (used by the stove while cooking).
  - Possible support a rating scale.
  - Possible support for comments on recipes.
  - Possible internationalization and support for English or metric units.
  - May integrate a time stamp for when meal was last cooked.
  - Consider support for sending recipes to other application users or exporting to common data formats and sending it via email.

### 3 Selected Screen Layouts

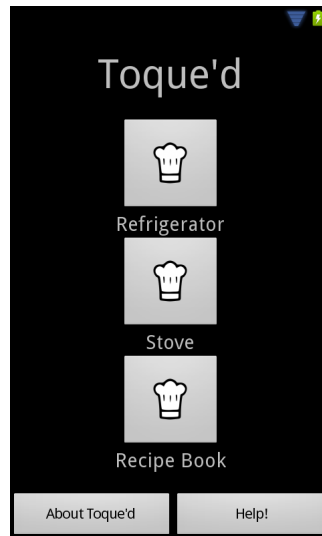


Figure 1: Main menu screen layout

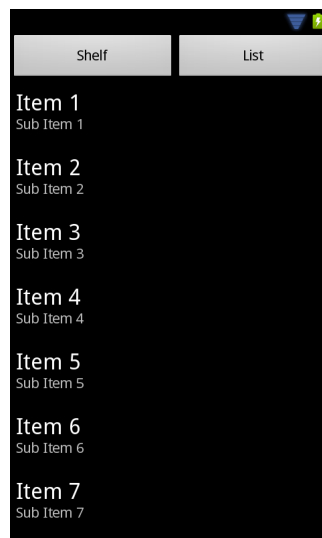


Figure 2: Refrigerator menu screen layout (shelf is for items in stock, list is for items needed)

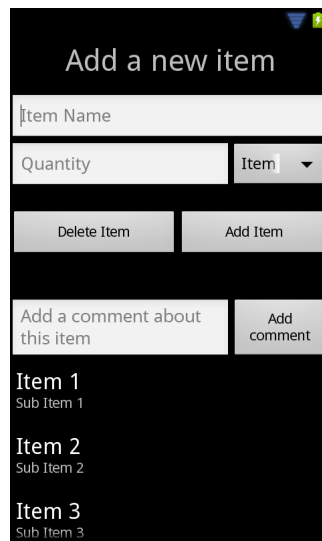


Figure 3: Fridge add/edit/delete item screen layout

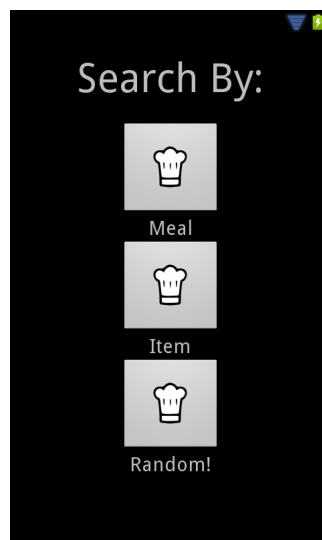


Figure 4: Stove menu screen layout (Hurray for code reuse!)



Figure 5: Recipe book menu screen layout (the pictures will be different at least)



Figure 6: Starting page for a recipe (links to next page)

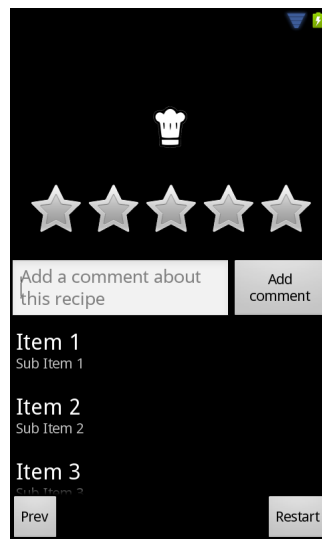
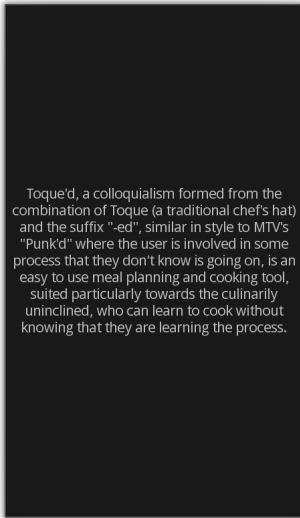


Figure 7: Ending page for a recipe (links to previous page and first page)



Figure 8: Help dialog box



Toque'd, a colloquialism formed from the combination of Toque (a traditional chef's hat) and the suffix "-ed", similar in style to MTV's "Punk'd" where the user is involved in some process that they don't know is going on, is an easy to use meal planning and cooking tool, suited particularly towards the culinarily uninclined, who can learn to cook without knowing that they are learning the process.

Figure 9: About dialog box



## 4 User Stories

1. Users will be able to use the application on a smartphone form factor so as to be able to view and create recipes on the go without having to carry around a tablet.
2. Users will be able to view common food items that they have in stock and that they have on their shopping list in order to know what items they have in stock and what they need to purchase for their meals.
3. Users will be able to add, edit, or delete items from both of these lists to reflect changes in current situation or different needs.
4. Users may be able to set low quantity alerts on items in the list, or even automatically have items added to the shopping list, so as to set personal reminders to acquire more of an item.
5. Users may be able to actively use a recipe and have the quantities in stock diminish according to the amount of each item used in the chosen recipe.
6. Users will be able to view recipes that they can cook from the ingredients they have in stock so as to see what recipes they can make.
7. Users will be able to search and filter these recipes by several criteria, possibly including: meal, ingredients, prep time, or rating, so they can better choose which recipe to use.
8. Users will be able to use a “random” function that selects a recipe at random from all possible recipes, so as to help indecisive or adventurous users. If no recipes are available from given ingredients, the system may choose a recipe with the fewest necessary ingredients and display that.
9. Users will be able to follow step by step recipes, bounded by a start page and an end page, so as to be able to quickly evaluate and then rate recipes.
  - On the start page, users will be able to view a photo of the finished product, see the prep time, ingredient list, and number of servings, as well as the overall rating, so as to be able to quickly assess the recipe.
  - On an individual recipe slide, users will be able to see a photo of the current step, the ingredients required for the step, and read the instructions associated with the step in order to make the recipe as simple as possible.
  - On the end slide, users may be able to rate the recipe as well as provide comments on the recipe, so as to provide feedback for next time.
10. Users may be able to integrate functions such as a count down timer into recipe slides, to further reduce the number of tools required to cook a recipe.

11. Users may have the option to switch between English and metric units so as to use recipes from other countries with ease and to aid with internationalization.
12. Users may be able to send recipes to other people using the application in a standard format so as to easily transfer knowledge and good food across the room or across the globe.

## 5 CRC Cards

These CRC card style bullets outline the main interactions between objects in the Toque'd project. The only part that I don't have fully planned out is the creation, population, editing, and deletion of recipe pages.

1.
  - Class: MainMenuActivity
  - Responsibility: Launches the main menu and the rest of the application with it. Imports recipes from files or the DB using other classes.
  - Collaboration: Works with all of the other classes either directly or indirectly through using an instance of the class by using an object of that class or by using an object that uses one of those classes. Links the model and the view by acting as the controller.
2.
  - Class: FridgeMenuActivity
  - Responsibility: Controls the view for the refrigerator. Switches between shelf and listviews.
  - Collaboration: Launches FridgeItemActivity to provide the ability to add, delete, or otherwise modify an item on the shelf or the list.
3.
  - Class: FridgeItemActivity
  - Responsibility: Is a close up view of an item on the shelf or on the list in the fridge. Users can add, delete, or edit items in either category. Additionally, it can store user comments about an item.
  - Collaboration: Pulls data in from FridgeItemStructure and is launched from FridgeMenuActivity.
4.
  - Class: StoveMenuActivity
  - Responsibility: Controls the main menu for the stove, which allows users to select between several buttons for different filtering options. Will send the user off to different other pages for filtering of recipes.
  - Collaboration: Links in with the StoveMealMenuActivity, StoveFilterMenuActivity, and StoveRandomActivity.
5.
  - Class: StoveMealMenuActivity
  - Responsibility: Allows a user filter recipes by meal: breakfast, lunch, dinner, dessert, or snack.
  - Collaboration: Launched from StoveMenuActivity, and directs a user to the RecipeBookViewActivity full of all of the recipes that apply to the chosen meal.
6.
  - Class: StoveFilterMenuActivity
  - Responsibility: Allows a user filter recipes by other metrics such as ingredients, prep time, or servings.

- Collaboration: Launched from StoveMenuActivity, and directs a user to the RecipeBookViewActivity full of all of the recipes that apply to the chosen filter quantities.
7.
    - Class: StoveRandomActivity
    - Responsibility: Supplies the user with a random recipe, chosen from the items on the shelf and the recipes in the instance of RecipeStructure.
    - Collaboration: Launched from StoveMenuActivity, and directs a user straight to a RecipeBookStartViewActivity for a randomly selected recipe.
  8.
    - Class: RecipeBookMenuActivity
    - Responsibility: Is the main menu for the recipe book, and provides an interface for users to add, edit, and delete recipes in the system.
    - Collaboration: Is launched from MainMenuActivity and can launch the add, edit, and delete views.
  9.
    - Class: RecipeBookViewActivity
    - Responsibility: Displays a list of selected recipes from the total list of recipes.
    - Collaboration: Is launched from the RecipeBookMenuActivity, StoveMealMenuActivity, or StoveFilterMenuActivity, and populated with recipes filtered from RecipeStructure.
  10.
    - Class: RecipeBookStartViewActivity
    - Responsibility: Displays the starting page of a recipe.
    - Collaboration: Is launched from a RecipeBookViewActivity of a StoveRandomActivity, and is the launching point for recipe instruction slides.
  11.
    - Class: RecipeBookInstructionActivity
    - Responsibility: Displays the current recipe instruction, photo, ingredients list, and any necessary documentation..
    - Collaboration: These are linked together between a RecipeBookStartViewActivity and a RecipeBookEndViewActivity.
  12.
    - Class: RecipeBookEndViewActivity
    - Responsibility: Displays the ending page of a recipe. Contains a rating for the recipe, a photo of the final product, and space for comments on the recipe.
    - Collaboration: Is the ending point in a recipe and is launched from the second to last slide in a series of recipe instructions.
  13.
    - Class: FileImport

- Responsibility: Imports a file from device internal storage or an SD card (if SD card setting is selected).
  - Collaboration: Is the superclass for RecipeImport and FridgeItemImport.
14. • Class: RecipeImport
- Superclass: FileImport
  - Responsibility: Imports the recipe files and photos. Either reads a .txt or .xml file or a SQLite DB and parses the information to populate data structures in the RecipeStructure class. Also pulls in comments about recipes.
  - Collaboration: Works with the RecipeStructure class to populate local structures that act as the model for the various recipe views.
15. • Class: FridgeItemImport
- Superclass: FileImport
  - Responsibility: Imports the refrigerator shelf and list items. Either reads a .txt or .xml file or a SQLite DB and parses the information to populate data structures in the FridgeStructure class. Also pulls in comments about food items.
  - Collaboration: Works with the FridgeStructure class to populate local structures that act as the model for the various shelf and list items.
16. • Class: RecipeStructure
- Responsibility: Provides setters and getters for different parts of different recipes. This is part of back end model of the application, and provides a fast and easy way to access photos and text in the different recipes.
  - Collaboration: Works with the RecipeImport class to populate the structures, and passes data to textviews and gets data from edittext fields.
17. • Class: FridgeStructure
- Responsibility: Will provide the backend for the listviews in the FridgeMenuActivity view.
  - Collaboration: Provides data for the FridgeMenuActivity and FridgeItemActivity and is populated through the FridgeItemImport class.

## 6 Sprints

I plan on breaking up the sprints into three sections, since there are three sprints. Sprint 1 will be front end development (View), Sprint 2 will be back end development (Model), and Sprint 3 will involve wiring them together and working out all the kinks (Controller + debugging), as well as implementing other lower priority features.

### 6.1 Sprint 1

For Sprint 1 I plan on accomplishing the following things and ending the sprint with a functional UI:

1. Create all the necessary view activities for the fridge, stove, and recipe book.
2. Add string, color, drawable, key, array, and other resources for the views, menus, and preferences.
3. Add listeners to all view activities and link through all pages properly.
4. Build up the necessary framework for menus and preferences in each of these, as needed.
5. Build up menus and preferences for those activities that could benefit from them (for instance, help and about on the main page).
6. Conduct some simple usability testing on the linked UI designs among friends and brothers to verify that the design is at least somewhat usable.
7. Try cooking one full meal using the phone in one hand, taking pictures, and reading a recipe from it (pull one off the internet, or prepopulate the views with dummy data).

Sprint 1 completed the fridge views, including population with data. Additionally, many of the string, key, array, and other resources were populated; however, this is always a work in progress. Listeners were added where appropriate, and every button and dialog/menu choice now results in an action. That being said, much of the actual content for recipes (including the recipe pages) doesn't exist. This is primarily due to uncertainty about the back end, which I realized would have to be created in parallel with the views, since the data stored and imported would change the view. I got some feedback about the UI's, primarily the fact that they were functional but not pretty. I'm not really sure what to do about that since nobody offered specific suggestions to fix that, but I will keep playing around with it. Sprint 1 wasn't difficult *per se*, but it was very time consuming. Additionally, I had to teach myself several new concepts (such as adapters), which took a good bit of time to test things out and then implement and debug.

## 6.2 Sprint 2

For Sprint 2 I plan implementing the back end (either a SQLite db or a file based system [like recipebook XML]) and then beginning to link up the recipes with their views:

1. Create simple test cases with both SQLite DB and an XML parser in an Android project.
2. Choose one of the two back end architectures and design the back end to store recipe data.
3. Choose one of the two architectures to store fridge item information. This may or may not be the same as the recipe storage back end.
4. Create start, page, and end recipe book views for displaying static recipes.
5. Create start, page, and end recipe book template views, and allow users to create new recipes.
6. Create editable start, page, and end recipe book template views, and allow users to edit their recipes, and then store them back in the back end.

I have two weeks (and a break) to implement this, so I will clean up some of the things from Sprint 1 and then continue on with Sprint 2.