***Things to Remember (main points from client comments on PDD):***
- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**
- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

---

# Software Requirements Specification

## for

# Cooking to Goal

**Version 1.0**

**Prepared by Ryan Spoon**

**Custom Fit Coding, LLC**

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

**March 17, 2011**

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and "circular statements" *(e.g. "We'll use a database to store data" is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- "will" … not "should" *(sound certain of what you're saying)*
- **Cut the "Fluff"**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

# Table of Contents

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and "circular statements" *(e.g. "We'll use a database to store data" is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- "will" … not "should" *(sound certain of what you're saying)*
- **Cut the "Fluff"**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

# Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
    - **spell check**
    - **ensure table of contents match the proper page numbers**

# 1. Introduction

## 1. Purpose

***This document is intended to act as an interface between the end user, client, and the development team, so that the user’s needs and the client’s desires, rather than the developer’s estimations, drive the development of the Cooking to Goal application through its first iteration release schedule.***

## 2. Document Conventions

***No specific document conventions used.  All information is presented in a single format.***

## 3. Intended Audience and Reading Suggestions

**This document is intended for the following audiences:**

***Carpe Aurum Venture Capital***
- ***CEO***
- ***Users***
- ***Business Manager***

***Custom Fit Coding, LLC***
- ***Project Manager***
- ***Developers***
- ***Testers***
- ***Documentation Writers***

***It is recommended that the reader of this document begin with the Introduction section, taking special note of the Purpose and Product Scope subsections.  The Overall Description section provides general information about the product, useful to any reader of the document.  Developers and Testers will find relevant information in the External Interface Requirements, while the System Features section will appeal to any reader interested in the software’s functionality.  Users will be interested in the User Interface and System Features.***

***Things to Remember (main points from client comments on PDD):***
- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**
- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

## 4. Product Scope

***The application will have an easy to navigate graphical user interface. It will be used to help customers create and compare meals and menus for a weekly period. Users will be able to choose from multiple menu options that will allow them to select a variety of meals while meeting their nutritional goals. A weekly shopping list for groceries will be generated based upon their meal selections.***

***A more detailed statement of scope for this project can be found on page 6 of the Cooking to Goal Software Project Management Plan.***

## 5. References

Ball, David. Cooking to Goal Software Project Management Plan. Version 2. Radford, Virginia: Custom Fit Coding, LLC; February 21, 2011.

# 2. Overall Description

## 1. Product Perspective

***The Cooking To Goal application is a new, self-contained product designed to assist users in shopping for meals related to their specific nutritional goals.***

## 2. Product Functions

***The primary functions of the application are:***
- ***To display recipes for meals via a graphical user interface (GUI) for easy selection***
- ***To add, edit, and remove recipes available for meal selections***
- ***To set weekly nutritional goals for the end user, family, and guests***
- ***To generate a week’s worth of meals with serving sizes scaled to meet the user’s weekly nutritional goals***

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

- ***To generate a shopping list for users to purchase ingredients for the weekly menu***

## 3. User Classes and Characteristics

***Power users***

***Users who work with the program on a very regular, almost daily basis. These users might make use of the more advanced or statistical program features. Examples include: fitness trainers, dietitians, body builders, and the extremely health-conscious consumer.***

***Casual users***

***The “core” users who work with the program occasionally, perhaps weekly. These users may be less concerned with advanced features of the application, but rely on the software to generate their weekly meal plan and shopping list. Examples include: the traditional “nuclear family”, including families or individuals on a diet.***

***Within the above groups, we also consider the following subgroups, based on technical ability:***

***Technical users***

***Those familiar with computers who will easily fall into regular usage of the program, based on prior experience with similar software or websites and their technical knowledge.***

***Non-technical users***

***Those who are less comfortable with computers who may require a “guiding hand” to help them navigate the software.***

## 4. Operating Environment

***Cooking to Goal will work on any operating system with a supported Java Runtime Environment 1.6.0 (Java SE 6 JRE) or higher installed. This includes most modern versions of Windows, Linux, and Mac OS.***

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
    - **spell check**
    - **ensure table of contents match the proper page numbers**

### 5. Design and Implementation Constraints

*The developer will consider the following limitations:*

- ***Must be programmed using only Java SE 6 technology.***
- ***The application does not require the use of databases to store information, but data design must be well formed to facilitate conversion to a database in the future.***
- ***Source code must be well documented both for maintenance and follow-on projects.***
- ***The application will employ the Model-View-Controller architecture, and adhere to proven design patterns.***

### 6. User Documentation

***The software will include built-in functionality for supplying users with graphical hints and tips on its usage. A one-page quick start guide will be provided to outline usage of the most common program features.***

### 7. Assumptions and Dependencies

***It is assumed that license and usage is and shall remain available for all of the software development tools and libraries required. These include, but are not limited to the Eclipse Integrated Development Environment, Git or other version control, a remote code repository service such as GitHub.com, Google Docs online office/collaboration suite, and third-party, open source APIs or libraries such as those covered in section 3.3.***

***It is also assumed that the end user will have a compatible Java Runtime Environment installed on a working computer.***

## 3. External Interface Requirements

### 1. User Interfaces

***Things to Remember (main points from client comments on PDD):***

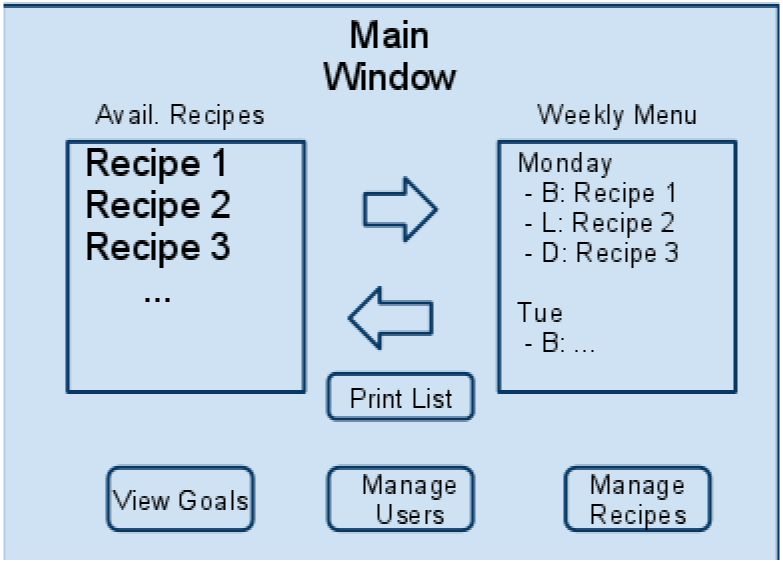
- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one."  *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

*A proposed example of the application’s graphical user interface:*

***Main Window:*** *The main window will be the entry point for the application.  The primary function of the application, Meal Planning, will be contained within this main window.  Users will select a recipe or meal item from a list, click an “add to menu” button (here represented as right-arrow) and the meal will be added to their weekly menu to a user-selected time slot for a given date/time.  To remove a meal, users will simply select the meal from the menu list and click the “remove” button (here, listed as a left-arrow).  A recipe’s user-supplied rating and nutritional facts will also be viewable from this window (TBD).*
*Users may also access the tertiary program features from this window, including printing their weekly shopping list, adding, editing or deleting users and recipes, or viewing their goals in the Goal Viewer.*



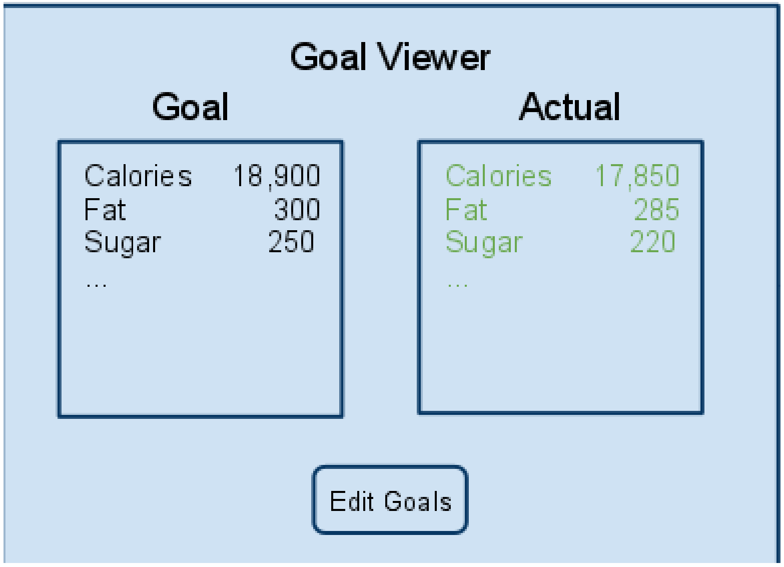
***Goal Viewer:*** *The goal viewer window will allow users to view the combined nutritional totals of all the users in the system, compared side-by-side with the nutritional values totalled from their weekly menu selections and scaled to fit their nutritional goals.  Users may also view and change their goals by clicking a button in this window which opens the User Manager window (see below).*

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**



| Goal | | Actual | |
|---|---|---|---|
| Calories | 18,900 | Calories | 17,850 |
| Fat | 300 | Fat | 285 |
| Sugar | 250 | Sugar | 220 |
| … | | … | |

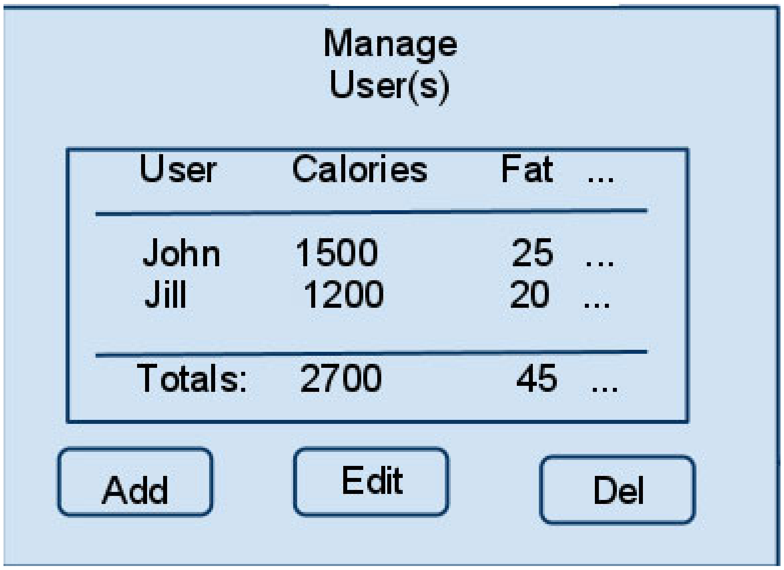
***User Manager:*** *The user manager window allows users to track new meal attendees as well as modify or remove existing attendees. An attendee is defined by their name and nutritional goals, which are viewable within the window.*

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**



| User | Calories | Fat | ... |
|---|---|---|---|
| John | 1500 | 25 | ... |
| Jill | 1200 | 20 | ... |
| Totals: | 2700 | 45 | ... |

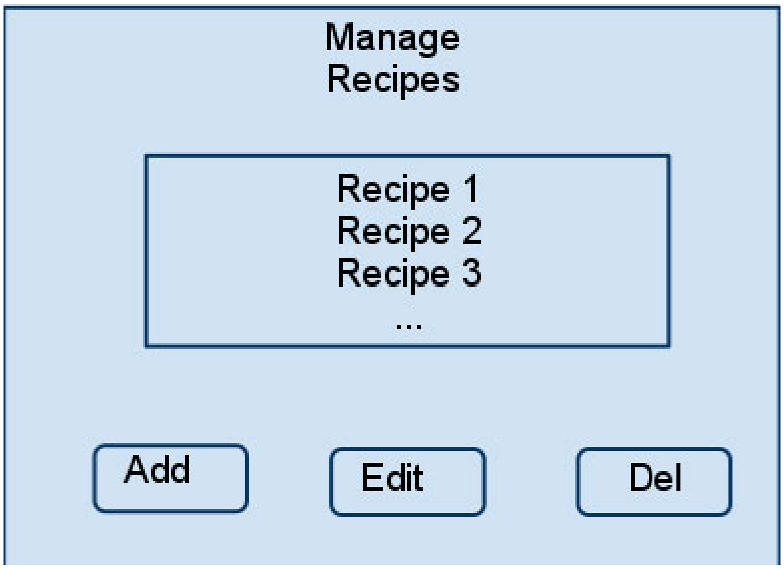
***Recipe Manager:*** *The recipe manager window allows users to enter new recipes into the system as well as modify or remove existing recipes. Users will also be able to view a recipe’s user-supplied rating and nutritional goals from this window (TBD).*

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**



***TBD GUI Elements:***

***Add/Edit Recipe:*** *A simple form for adding or editing recipes*

***Add/Edit User:*** *A simple form for adding or editing users*

***Recipe Nutritional View****: Will resemble the familiar FDA standard food nutrition label*

## 2. Hardware Interfaces

***The application will support the standard hardware interface of monitor, mouse, keyboard, and printer. The visual representation of the program will display on any standard monitor. Most interaction with the software will take place via usage of a mouse, but will require the keyboard for entering user and recipe information. Data will be stored to the local hard drive of the end-user’s computer. The application must have access to a printer in order to use the printing functionality.***

## 3. Software Interfaces

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

***The application will primarily utilize the built-in tools of the Java 6 standard library. A few third-party libraries and APIs are currently under consideration.*** ~~Use nothing which creates licensing issues when the product is distributed to our customers~~ ***These include, but are not limited to:***

- ***Standard Widget Toolkit (SWT) -*** *a graphical widget library (license: CPL)*
- ***Jasper Reports -*** *a reporting API which write to screen or a printer, and also has functionality to export reports to various popular file formats (PDF, Excel, XML, etc.) (license: LGPLv3)*
- ***JDOM -*** a Java representation of an XML document (license: Apache without acknowledgment clause)

***Licensing for all third-party libraries and APIs will be verified for acceptable use before inclusion.***

## 4. Communications Interfaces

**No communication interfaces will be necessary.**

# 4. System Features

## 4.1 User Maintenance Screen
## 4.2 Meal Planner
## 4.3 Manage Recipes in the Recipe Box
## 4.4 View Nutritional Goals
## 4.5 Print Shopping List
## 4.6 Printable Recipes

## 4.1 User Maintenance Screen

### 4.1.1 Description and Priority

The User Maintenance screen is the container for user details such as name, age, nutritional goals, etc. Users may add, edit, or delete their profiles from within

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and "circular statements" *(e.g. "We'll use a database to store data" is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- "will" … not "should" *(sound certain of what you're saying)*
- **Cut the "Fluff"**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

this feature. Upon the first initial launch of the program, if a user does not wish to enter their information immediately, a default profile is loaded so that the user may explore the system immediately. The user may claim the default profile at any time, as well as add additional profiles for family members or guests who also have nutritional goals.

**Priority:** High

### 4.1.2 Stimulus/Response Sequences

| Stimulus | Response |
|---|---|
| Initial launch of program | User is prompted to edit/enter information such as name, age, and nutritional goals. |
| Use selects "User Maintenance" option from main window | User is prompted to add, edit, or delete new users with information such as name and nutritional goals. |
| User selects Add User from User Maintenance window | User is prompted to fill in name and nutritional goals for a new user |
| User selects Edit User from User Maintenance window | User is prompted to edit the currently selected user's name and nutritional information |
| User Selects Delete User from User maintenance window | User is prompted to confirm deletion of the selected user |
| User declines to enter information at initial program launch | A predefined Default user profile is loaded. |

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- “Forgive the long letter, I didn't have time to write a short one.” *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
    - **spell check**
    - **ensure table of contents match the proper page numbers**

### 4.1.3 Functional Requirements

a. MealPlanner confirms that user information exists
b. Load GUI with user information
c. GUI displays edit user information window
d. User enters information, such as name and nutritional goals
e. User clicks “save” button
f. MealPlanner saves information to file
g. GUI closes information window
h. Variation 1: User information does not exist at step a
    i. GUI loads default user information
    ii. continue from step c
i. Variation 2: User clicks “cancel” button at step e
    i. MealPlanner confirms that user information exists
    ii. Continue from step g
    iii. Variation 1: User information does not exist at step i
        1. GUI displays “using defaults” message.
        2. Continue from step f
j. Variation 3: User clicks “Add User” button
    i. GUI displays an add user form with fields for name and nutritional goals
    ii. Variation 3.1: User enters information and clicks “save”
        1. Program adds user to the list of users
    iii. Variation 3.2: User clicks “cancel” at any time
        1. Return to step a
k. Variation 4: User clicks “Edit User” button
    i. GUI displays an Edit user form with populated fields for name and nutritional goals for the currently selected user
    ii. Variation 4.1: User enters information and clicks “save”
        1. Program edits the select user’s profile
    iii. Variation 3.2: User clicks “cancel” at any time

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
    - **spell check**
    - **ensure table of contents match the proper page numbers**

        1. Return to step a

    l. Variation 5: User clicks “Delete User” button

        i. GUI displays a “confirm deletion” dialog box

        ii. Variation 3.1: User enters information and clicks “confirm”

            1. Program deletes currently selected user from the list of users

        iii. Variation 3.2: User clicks “cancel” at any time

            1. Return to step a

## 4.2 Meal Planner

### 4.2.1 Description and Priority

The Meal Planner screen is the heart of the application. From here, the user may select from predefined recipes to add to their weekly meal plan or choose to add their own recipes to the system. As meals are added to the weekly meal plan the application will calculate the necessary serving sizes to stay within the designated nutritional goals.

**Priority:** High

### 4.2.2 Stimulus/Response Sequences

| Stimulus | Response |
|---|---|
| Initial launch of program | System loads predefined meal list. |
| User selects a meal from the meal list to add to their weekly meal selection | User is prompted to select which day and meal time slot to add the meal (ie. Breakfast, Lunch, Dinner, Snack, etc.) |
| User removes a meal from their weekly plan | Meal is removed from weekly plan |

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
    - **spell check**
    - **ensure table of contents match the proper page numbers**

| | |
|---|---|
| User chooses to manage the recipes in the meal list | Program displays Manage Recipes window (see section 4.3) |

### 4.2.3 Functional Requirements

a. MealPlanner confirms that meals exist
b. Load GUI with meal information
c. GUI displays meal planning window
d. Variation 1: User adds a meal to their weekly menu
    i. User selects a meal or recipe from the list
    ii. User clicks the “add to menu” button to move the item to their weekly menu
    iii. User is prompted to select a time slot on the weekly menu
    iv. Selected meal is added to weekly menu
    v. return to step c
e. Variation 2: User removes a meal from their weekly menu
    i. User selects a meal item from their weekly menu
    ii. User clicks the “remove from menu” button to remove the item from their weekly meal list
    iii. return to step c

## 4.3 Manage Recipes in the Recipe Box

### 4.3.1 Description and Priority

The Manage Recipes feature allows users to enter their own recipes to the meal selection menu as well as edit and delete recipes from the meal list. Recipes will include a name, the nutritional information for the prepared recipe, ingredients, cooking instructions, serving size, and a user rating.

**Priority:** High

### 4.3.2 Stimulus/Response Sequences

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
    - **spell check**
    - **ensure table of contents match the proper page numbers**

| Stimulus | Response |
|---|---|
| User opens Manage Recipes window | Program displays a list of modifiable recipes with buttons to add, edit, and delete recipes |
| User chooses to add a new recipe to the meal list | Program displays a recipe entry window |
| User chooses to edit a recipes in the meal list | Program displays a recipe edit window, populated with the currently selected recipe’s information. |
| User chooses to delete a recipe from the meal list | Program confirms user’s deletion selection, then deletes the recipe from the system. |

### 4.3.3 Functional Requirements

a. RecipeManager confirms recipes exist in the system
b. Load GUI with list of recipes and add/edit/delete buttons
c. Manage Recipe GUI displays
d. Variation 1: User selects Add Recipe
    i. Add Recipe window is displayed with fields for ingredients, instructions, and nutritional information
    ii. User clicks “save” button
    iii. New recipe is added to system
    iv. Variation 1.1: User clicks “cancel” button
        1. Program returns to step a
e. Variation 2: User Selects Edit Recipe
    i. Variation 2.1: Load information for selected recipe
        1. Edit Recipe window is displayed with fields for ingredients, instructions, serving size, and nutritional information for the selected recipe
    ii. Variation 2.2: User failed to select a recipe before clicking button

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

        1. Warn user and return to step e
    f. Variation 3: User Selects Delete Recipe
        i. Variation 3.1: Load information for selected recipe
            1. Delete confirmation window is displayed with Confirm and Cancel Buttons
            2. Variation 3.1.1: User clicks Confirm
                a. Recipe is deleted from the system
            3. Variation 3.1.2: User clicks Cancel
                a. Return to step c
        ii. Variation 3.2: User failed to select a recipe before clicking button
            1. Warn user and return to step f

~~**Can we have ability to "rate" recipes? Yes, but with limited functionality (ryan)**~~

**Client will supply desired standard content for recipes**

**Where is "Fit Menu to Nutritional Constraints"?**

*(My assumption is that this is done automatically behind the scenes. A user may view the scaled menu within the Goal Viewer window (see interface section). Should the user have to initiate the scaling function themselves ?? -- we’ll need to add a button for that, but the internals wouldn’t change -- Drew see below and review:)*

## 4.4 View Nutritional Goals

### 4.4.1 Description and Priority

*The program provides the ability to view the combined nutritional goals of all of the users in the system, side-by-side with the scaled nutritional totals of their weekly meal selections.*

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- “Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

**Priority:** Medium

### 4.4.2 Stimulus/Response Sequences

| Stimulus | Response |
|---|---|
| Use selects View Goals from main window | Program opens Goal Viewer window with a Goal vs Actual display of nutritional facts. |
| User selects Edit Goals from Goal Viewer window | Program opens User Manager window, where user may edit nutritional goals |

### 4.4.3 Functional Requirements

a. Variation 1: User selects View Goals from main window
   - i. Program totals combined goals of all users in the system
   - ii. Program totals combined nutrition facts of weekly menu items
   - iii. Program scales weekly menu totals to fit combined nutritional goals
   - iv. Load GUI of Goal Viewer window with display of Goal vs Actual nutrition information.

b. Variation 2: User selects Edit Goals from Goal Viewer window
   - i. Program loads User Manager window (see section 4.2)

## 4.5 Print Shopping List

### 4.5.1 Description and Priority

Based on the user’s weekly meal selection, the program will generate and print a weekly shopping list. The list will contain all required ingredients and the weekly total quantity for each ingredient. ~~Should we allow user to input planning period instead of fixing it at one week? NO (ryan)~~

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

**Priority:** High

### 4.5.2 Stimulus/Response Sequences

| Stimulus | Response |
|---|---|
| User Selects Print Shopping List option | System displays a preview of the week’s shopping list with a print button to select printer options. |

### 4.5.3 Functional Requirements

a. Variation 1: User selects Print Shopping List
    i. Confirm that a weekly meal list exists
    ii. Variation 1.1: Weekly meal list does not exist
        1. Issue an error, and return to main menu
    iii. Variation 1.2: Weekly meal list does exist
        1. Compute the week’s shopping list from the weekly meal plan
            a. Program totals combined nutrition facts of weekly menu items
            b. Program scales weekly menu totals to fit combined nutritional goals
        2. Display a preview of the shopping list with a Print and Cancel button
        3. Variation 1.2.1: User clicks Print button
            a. Determine if a printer exists
                i. Variation 1.2.1.1: Printer does not exist
                    1. Issue an error and return to step a
                ii. Variation 1.2.1.2: Printer does exist
                    1. Display print selection dialog
                    2. Return to main menu when print dialog closed
        4. Variation 1.2.2: User clicks cancel button
            a. Return to main menu

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

## 4.6 Printable Recipes

### 4.6.1 Description and Priority

The application will include the ability to print recipes in a common, familiar format.

**Priority:** Low

### 4.6.2 Stimulus/Response Sequences

| Stimulus | Response |
|---|---|
| User Selects Print Recipe option | System displays a preview of the selected recipe with a print button to select printer options. |

### 4.6.3 Functional Requirements

a. Variation 1: User selects Print Recipe
    i. Display a preview of the recipe with a Print and Cancel button
    ii. Variation 1.2.1: User clicks Print button
        1. Determine if a printer exists
            a. Variation 1.2.1.1: Printer does not exist
                i. Issue an error and return to step a
            b. Variation 1.2.1.2: Printer does exist
                i. Display print selection dialog
                ii. Return to main menu when print dialog closed
    iii. Variation 1.2.2: User clicks cancel button
        1. Return to main menu

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

*<< Use this format for additional features >>*

## 4.n Feature Name

### 4.n.1 Description and Priority

*<Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>*

**Priority:** High/Med/Low

### 4.n.2 Stimulus/Response Sequences

*<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>*

| Stimulus | Response |
|---|---|
| Use does this... | Program does this... |
| User does that... | Program does something else.... |
| etc... | etc... |

### 4.2.3 Functional Requirements

*<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use “TBD” as a placeholder to indicate when necessary information is not yet available.>*

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

c. First user does this
d. Then something happens
e. Then user does something else
f. Then something else happens
g. Variation 1: Something different happened at step ‘b’
    i. Something else
    ii. continue from step c

*<< end format >>*

# 5. Other Nonfunctional Requirements

## 1. Performance Requirements

***All user actions will result in a visible screen response within 1 second.***

## 2. Safety Requirements

***There are no specific safety requirements.***

## 3. Security Requirements

***There are no specific security requirements.***

## 4. Software Quality Attributes

**Usability:** The application will have a simplistic interface which will be widely accessible to a variety of users, from those will minimal computer experience to seasoned pros.  Intuitive interface

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and "circular statements" *(e.g. "We'll use a database to store data" is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- "will" … not "should" *(sound certain of what you're saying)*
- **Cut the "Fluff"**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

design will lead the user towards the tasks they wish to complete easily and efficiently.
**Maintainability:** Use of object oriented best practices and the application of design patterns and the model-view-controller architecture will facilitate a code base which is easy to modify and maintain.
**Reliability:** The simplicity of the application, coupled with the use of Java SE's secure data encapsulation practices and error management, the application will not suffer from memory leaks typical of other platforms.
**Testability:** Object oriented coding and unit testing will enable the development team to easily test, modify, and retest manageable units of code quickly and individually, leaving the rest of the code base unharmed. JUnit will be used by the development team to conduct unit testing.

## 5. Business Rules

***User authentication is not required; therefore only one role will exist.***

# 6. Other Requirements

***Provide useful documentation and ensure that core functionality is not tied to interface implementation.***

# Appendix A: Glossary

<u>**Technical Terms:**</u>
***Graphical User Interface:*** *(**GUI**, pronounced "gooey") The visual representation of a program which allows a user to interact with the program through images rather than pure text.*
***User:*** *Any person or persons who will utilize the program.*
***Software / Application / Program:*** *In this document, The Cooking to Goal computerized meal planner system.*
***Database:*** *A system intended to organize, store, and retrieve large amounts of data easily.*
***Source / Source Code:*** *The set of instructions programmers write which are interpreted by the computer to perform the desired functions.*
***API / Library:*** *Pre-built bundles of code, useful for speeding up software development.*

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

**Nutritional Terms:**

***Meal:*** *A combination of recipes. Dinner would be considered a meal that may contain the recipes for preparing steak, green beans, and garlic bread.*

***Recipe:*** *A set of ingredients and instructions for making or preparing a food dish as part or all of a meal. Recipes can be considered full meals and can be as simple as a name and estimated nutritional information for recipes not made from scratch (e.g. "Neighbor's Cook-Out" 1500 cal, 20g fat, 10g protein). Recipes can be rated and set to not allow serving size scaling by the nutritional goal calculator.*

***Ingredient:*** *The most basic component of a recipe. If a recipe is the ingredient for another recipe (e.g. pie crust for apple pie) the two recipes can be grouped together within a meal to show their relationship. In other words, you will not be able to list the pie crust as a recipe within the apple pie recipe. Recipes may list the same material more than once if the same item is used more than once in the same recipe.*

***Nutritional Goal:*** *A specified amount of nutrition, such as calories, fat, and cholesterol, that the user does not want to exceed or go below.*

# Appendix B: Analysis Models

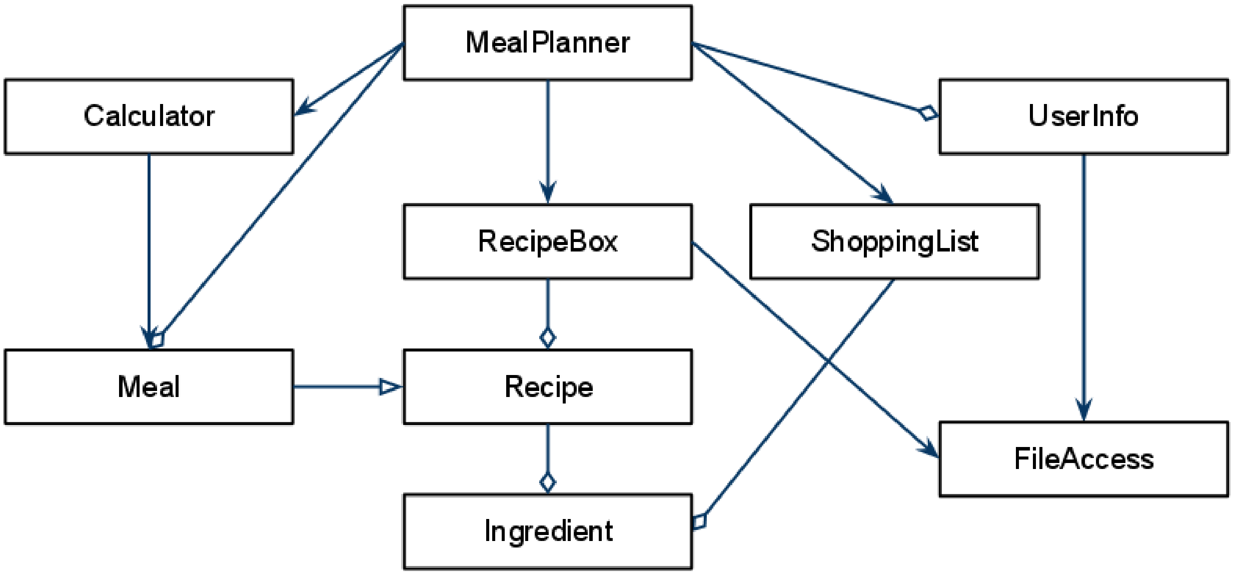
Below is a class relationship diagram. Please note that as we design the application some of these classes and relationships may change.

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**



Below is an example interface flow diagram:

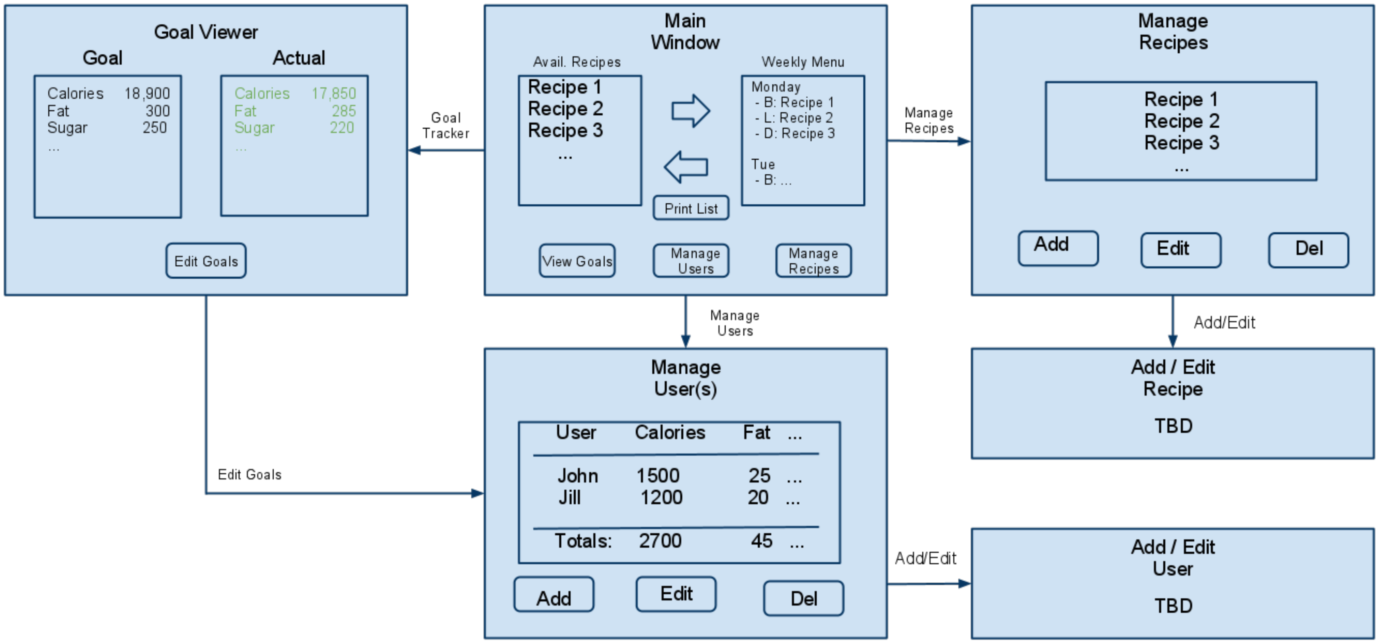
***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and "circular statements" *(e.g. "We'll use a database to store data" is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- "will" … not "should" *(sound certain of what you're saying)*
- **Cut the "Fluff"**

**To Do:**

- ~~**Finish Software Interface**~~
- ~~**In Features: finish Stimulus/Response Sequences and Functional Requirements**~~
- ~~**Finish User Interface**~~
- ~~**Finish Software Quality Attributes**~~
- ~~**Add any additional terms to the glossary**~~
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**

GUI Layout Flow
for Cooking To Goal Version 1.0



Scaling Function View (**do not need - done automatically?? - drew_** and ~~Plan vs. Goal Display?~~

# Appendix C: To Be Determined List

***3.1 - Main Window:*** *A recipe's user-supplied rating and nutritional facts will also be viewable from this window (TBD).*

***3.1 - Recipe Manager:*** *Users will also be able to view a recipe's user-supplied rating and nutritional goals from this window (TBD).*

***3.1 - TBD GUI Elements:***

- ***Add/Edit Recipe:*** *A simple form for adding or editing recipes*
- ***Add/Edit User:*** *A simple form for adding or editing users*
- ***Recipe Nutritional View****: Will resemble the familiar FDA standard food nutrition label*

***Things to Remember (main points from client comments on PDD):***

- Eliminate redundancy and “circular statements” *(e.g. “We’ll use a database to store data” is redundant because storing data is obviously what a database does)*
- "Forgive the long letter, I didn't have time to write a short one." *(Say a lot with few words)*
- be Clear & Concise *(keep things simple)*
- “will” … not “should” *(sound certain of what you’re saying)*
- **Cut the “Fluff”**

**To Do:**

- **~~Finish Software Interface~~**
- **~~In Features: finish Stimulus/Response Sequences and Functional Requirements~~**
- **~~Finish User Interface~~**
- **~~Finish Software Quality Attributes~~**
- **~~Add any additional terms to the glossary~~**
- **QA verify**
  - **spell check**
  - **ensure table of contents match the proper page numbers**