|  |
| --- |
| Regis University |
| Recipe Manager |
| Design document |
|  |
| **Peter Varner-Howland** |
| **5/8/2013** |

|  |
| --- |
| CERTIFICATION OF AUTHORSHIP  I certify that I am the author of this paper and that any assistance I received in its preparation is fully acknowledged and disclosed in the paper. I have also cited any sources from which I used data, ideas, or words, either quoted directly or paraphrased. I also certify that this paper was prepared by me specifically for this course.  Peter Varner-Howland |

Contents

[Revision History 2](#_Toc357260241)

[Introduction and Business Case 2](#_Toc357260242)

[Use Cases 3](#_Toc357260243)

[Use Case 1A 4](#_Toc357260244)

[Use Case 1B 5](#_Toc357260245)

[Domain Layer 6](#_Toc357260246)

[Class Diagram 6](#_Toc357260247)

[Service Layer 7](#_Toc357260248)

[Class Diagram 7](#_Toc357260249)

## Revision History

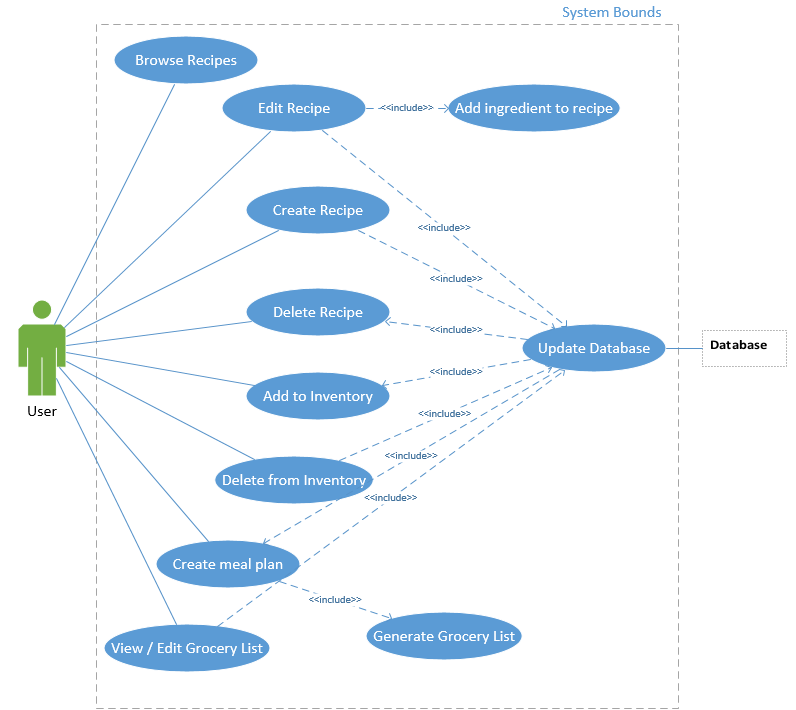
|  |  |  |  |
| --- | --- | --- | --- |
| **Revision Number** | **Date** | **Author** | **Description** |
| 1 | 5/15/2013 | Peter Varner-Howland | Added Domain Layer description and diagram |
| 2 | 5/25/13 | Peter Varner-Howland | Added Service Layer description and diagram  Updated Domain diagram to include RMObject inheritance |
| 3 | 5/26/13 | Peter Varner-Howland | Updated Domain diagram with peer suggestions and corrections  Added an explanation of RMObject to the domain intro  Corrected typo in Use Case diagram |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Introduction and Business Case

Anyone who has ever cooked for a family knows how taxing it can be to keep track of all the variables: What are we going to eat? What food do we have? What do I need from the grocery store? How much? This is just the beginning of the myriad of components that go into planning a family’s meals. The main objective of this application is to focus on the issue of assembling a grocery list, which will solve many other issues at the same time.

The ideal way this app would be used is simple; the user would first enter all of their family’s favorite recipes (or new ones they want to try). The information provided in the recipe will include necessary ingredients and their amounts, along with appropriate sides for this meal (the sides would be housed as separate recipes). All of these recipes would be stored so that the user may browse / edit them, as well as used to create a weekly meal plan. Second, the app would maintain a database of all foods currently owned, to which the user can add / remove from at any time. Both of these features would then facilitate the final feature: meal planning. The user may initiate a weekly meal plan by selecting any number of new or pre-entered recipes and sides. The app will then generate a grocery list which will take into account what foods are needed for the selected meals and the food currently owned. Once the list is generated, the user may then edit said list to add or remove any item they may or may not need. The final list can then be exported as a text file, uploaded to a cloud drive, emailed, printed, etc.

## Use Cases



### Prioritized Use Cases

1. Update Database
2. Create Recipe
3. Add Ingredient to recipe
4. Browse Recipes
5. Edit Recipe
6. Add to Inventory
7. Delete from Inventory
8. Create Meal Plan
9. Generate Grocery List
10. View / Edit Grocery List
11. Delete Recipe

### Use Case 1A

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case ID:** | 1A | | |
| **Use Case Name:** | Update Database | | |
| **Created By:** | Peter Varner-Howland | **Last Updated:** |  |
| **Date Created:** | 5/10/2013 | **Date Last Updated:** |  |
|  |  |  |  |
| **Actors:** | User | | |
| **Description:** | Creates a connection to a remote database and adds an entry to the appropriate table. | | |
| **Trigger:** | User creates, edits, or deletes items from their inventory or a recipe from the list. | | |
| **Preconditions:** | User clicks save to save a recipe (either creating or editing); user clicks Delete; user clicks add or delete in the inventory screen. | | |
| **Post conditions:** | User is notified that the information is saved. | | |
| **Normal Flow:** | 1. User clicks the trigger button. 2. A connection to the database is established. 3. An entry is added, edited, or removed from the database. 4. Success notification. | | |
| **Alternative Flows:** | NA | | |
| **Includes:** |  | | |
| **Exceptions:** | 1. Database connection failed | | |
| **Priority:** | High | | |
| **Frequency of Use:** | Up to 20 times / minute | | |
| **Special Requirements:** |  | | |
| **Assumptions:** | User has an internet connection. | | |
| **Notes and Issues:** |  | | |

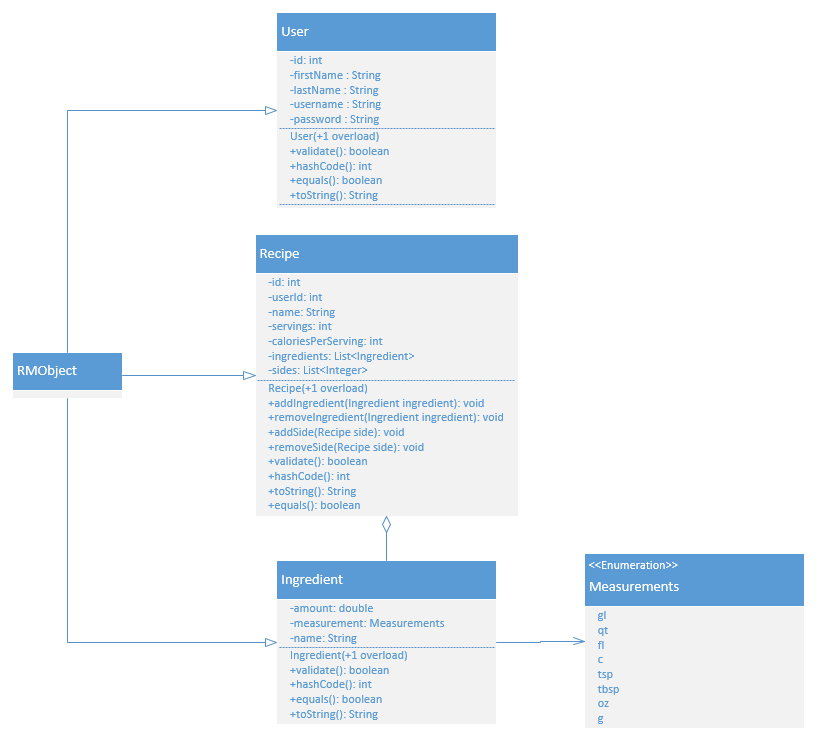
### Use Case 1B

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case ID:** | 1B | | |
| **Use Case Name:** | Create Recipe | | |
| **Created By:** | Peter Varner-Howland | **Last Updated:** |  |
| **Date Created:** | 5/10/2013 | **Date Last Updated:** |  |
|  |  |  |  |
| **Actors:** | User | | |
| **Description:** | Allows user to create and save a recipe for later use. | | |
| **Trigger:** | User clicks "Create Recipe" | | |
| **Preconditions:** | NA | | |
| **Post conditions:** | User is notified that the recipe has been saved. | | |
| **Normal Flow:** | 1. User clicks "Create Recipe" either from the home screen, or from the menu bar. 2. User gives the recipe a title. 3. User can add any number of **ingredients**. 4. User clicks "Save". 5. User is notified that the recipe has been saved. | | |
| **Alternative Flows:** | NA | | |
| **Includes:** | Add Ingredient to recipe | | |
| **Exceptions:** | NA | | |
| **Priority:** | Normal | | |
| **Frequency of Use:** | Once every ~20 minute | | |
| **Special Requirements:** |  | | |
| **Assumptions:** | User has an internet connection. | | |
| **Notes and Issues:** |  | | |

## Domain Layer

The domain layer consists of three different classes: User, Recipe, and Ingredient; it also contains an enumeration containing all types of measurements (tsp., tbsp., oz., etc.). These classes will serve as a bridge between the database and the application; for instance, whenever a recipe is accessed from the database, it will be stored in memory as a Recipe object. Data can then be added, removed, or edited on these objects by the various services, then stored back in the database. Due to this lack of distinct functionality, no unit tests are required at this time. Each domain class inherits from a base class (RMObject) to allow data access methods in the service layer to be more generic.

### Class Diagram

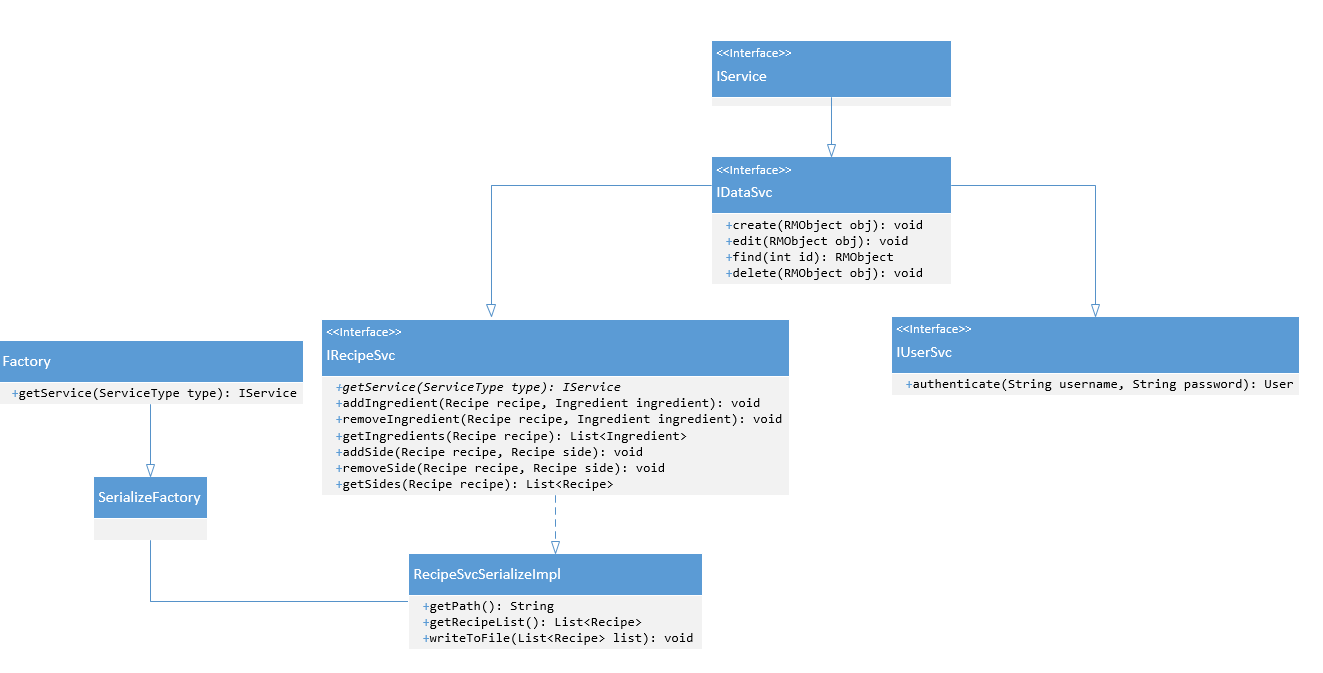


## Service Layer

The Service Layer consists of a series of interfaces all of which inherit from IService. The IDataSvc contains all methods which deal with data storage and retrieval (create, edit, find, remove). The IRecipeSvc contains all methods which deal with editing and listing ingredients and sides on specific recipes. The IUserSvc deals exclusively with user authentication. The current interface implementation uses serialization to fulfill all of the data storage / retrieval methods; usage of a database will come later.

An abstract factory class will serve as the means of instantiating the service implementations. Currently there is only one factory type, but with the addition of multiple service implementation types, more factory types will be added.

### Class Diagram

\*inherited methods are omitted in child classes / interfaces.