**ChefsHub: A recipe sharing platform for culinary enthusiasts.**

**Requirements Document**

Dr. Chính Hoàng

Thursday, Nov. 27, 2014

Group #4 (Ryan Burke, Ranadeep Polavarapu, Lee Glendenning, Bruno Salapic)

****

Table of Contents

Introduction 3

1. Industry Overview 3

1.1 Background 3

1.2 Glossary 4

1.3 Core Value Proposition 4

2. Product Overview 4

2.1 General Description 5

2.2 Goals and Objectives 5

2.3 Identification of ChefsHub’s Needs 6

3. System Users (Actors) 6

3.1 Actor Descriptions 6

3.1.1 User 6

3.1.2 Corporate 6

4. Initial Use Case Identification 8

4.1 ChefsHub Cooking Application System Wide Use Case Diagram 8

4.2 Identify Scenarios 9

4.2.1 SortRecipeByCuisine scenario 9

4.2.2 DeleteRecipe scenario 9

4.2.3 AddRecipe scenario 9

4.2.4 RateRecipe scenario 10

4.2.5 SortRecipeByPreparationTime scenario 11

4.2.6 SortRecipeByDifficulty scenario 11

4.2.7 SortRecipeByDate scenario 11

4.2.8 LikeRecipe scenario 12

4.2.9 UnlikeRecipe scenario 12

4.3 Identify Use Cases 13

4.3.1 sortByCuisine use case 13

4.3.2 AddRecipe use case 13

4.3.3 DeleteRecipe use case 14

4.3.4 RateRecipe use case 14

4.3.5 SortByPreparationTime use case 15

4.3.6 SortByDifficulty use case 16

4.3.7 SortByDate use case 16

4.3.8 LikeRecipe use case 17

4.3.9 UnlikeRecipe use case 18

5. Non-Functional Requirements 18

Requirements

# Introduction

Introducing **ChefsHub**, a platform and network where those who are passionate about cooking can meet and share recipes with one another.

# 1. Industry Overview

## 1.1 Background

The cooking industry is a multi-billion dollar industry. It is an industry that is composed of the masses in the world, from professional chefs to everyday regular moms. Food is a basic necessity for human survival and it is what sustains us on a day-to-day basis. Canadians spent more than $92 billion alone on food and beverages in 2007.[[1]](#footnote-1) The food and beverage industry is an important component in any country’s economy. Nowadays the organic food industry is booming as well since people are becoming more health conscious by eating food that is healthy and doesn’t contain any external additives like hormones or preservatives. Tastes are getting more diverse as countries around the world are becoming more multicultural and incorporate cuisines from a wide variety of different cultures.

People are always looking for ways to get the best recipes from several cuisines. We decided to cater to the people’s needs by creating a mobile application that will make searching and sharing recipes with the rest of the world.

In terms of business and profitability, the spending on the food and beverage industry has displayed a drastic growth from 1961 to 2008 as portrayed in a Statistics Canada report. [[2]](#footnote-2) As for our application, we can monetize this in the future by placing ads. The application itself will be free to use, however we can tie up with companies that make cooking accessories and advertise their products via our app.

To learn more about the cooking industry, our group researched by using statistics Canada.

## 1.2 Glossary

**Cooking**: The act of food preparation, which is often considered an art by many people around the world. Different techniques and ingredients were incorporated to modify the taste of the food that was consumed throughout the millions of years that humans have existed.

**User**: Is any external entity (a person) that is using our application regardless of whether they are logged in or not.

**Recipe**: A recipe is a set of detailed instructions that outline how to prepare or make a culinary dish.

**Like**: A *like* in our application is the user’s fondness of a recipe and a way to store favourite recipes.

**Dislike**: A *dislike* in our application is when the user is no longer fond of a certain recipe and wishes to no longer see this recipe in his liked list.

**Rate**: A form of expressing the user’s evaluation or assessment of the culinary dish that the recipe is describing.

**Sorting**: Sorting is a feature in our application where the *User* can sort by certain filters or options such as Recipe difficulty, cuisine, preparation time, and date published.

## 1.3 Core Value Proposition

Our team has performed through research of the food and beverage industry and after careful consider we have come up with a strict CVP (Core Value Proposition) to outline the core benefit our business provides to the general public (the masses, since everyone in the world cooks food or makes beverages of some sort).

*“To provide a FREE medium of exchange where the public can share their love for the culinary art of cooking.”*

# 2. Product Overview

## 2.1 General Description

The goal of our software is to be a social platform like *Facebook* (a general social platform) or *GitHub* (a social hub to share open source software) but cater our primary focus towards the food and beverage industry. This means that we wish to be a social hub where people enthusiastic and passionate about cooking can come and share their culinary delights and interact with others who share the same ideals, principles, and passion about cooking.

**The ChefsHub Cooking Application will provide primary stakeholders with the ability to share their culinary passion by:**

* Offering the **User** the ability to share his/her culinary creations in the form of recipes for the world to see.
* The **User** has the ability to sort by several different options such as cuisine, preparation time, and etc.
* Supplying **Corporate** (Administrators) with ability to moderate content by disableusers’ accounts if their actions are deemed to be abusive to user environment.

This system will be designed in such a way that it will be heavily scalable in future with little to no tweaks in terms of core architecture and technology used as it is the latest and most up to date web architecture. This will allow the company to flourish should we wish to monetize this application in the future by adding advertisements to generate a some form of revenue to function as a real-time profit generating company.

Our product will better the current high demand for sharing food and beverages in pictographic and text format. There is already a huge market for people taking picture of their food prior to eating and uploading it on popular image sharing websites (Ex. Twitter, Facebook, and Instagram [the most prominent food sharing application currently]). Thus, we wish to capture these ***foodies*** and provide them with an ability to share their food and beverage images and their own culinary dishes by publishing recipes. We feel as if we are at an advantage here because the target market is so large and limitless since **EVERYONE** is a *foodie* of some sort.

## 2.2 Goals and Objectives

The agreed upon goals and objectives by our team were:

* Provide a platform for sharing recipes.
* Create an online experience the users cherish and cause them to return several times throughout the day. In a way, it is like checking your phone for text messages or Facebook messages, we want the same level of commitment and interaction.
* Provide the ability to show satisfaction and dissatisfaction of recipes by the users.
* Reduce the time spent by users for searching for recipes, we will offer a quick and easy search that will generate a list of related matches to the user provided search keyword.

## 2.3 Identification of ChefsHub’s Needs

We have determined that ChefsHub has certain needs but as in reality each *need* has a corresponding *want*.

* **Wants:** We want users to sign up prior to using.
* **Needs:** To have users login to nudge them in the direction of sharing content and becoming truly “involved” in our product.

**Rationale**: We determined there is a divide here and a possible *flaw* or *issue* that might cause certain users to become dissatisfied if we forced sign in upon them. We have seen certain applications fail because they forced registration and signup down users’ throats and it all went down hill from that point on. We have come up to the equilibrium point where we will allow users to view recipes publicly without sign in or a registered account however, we will limit the recipe interaction such as rating, liking, and creating to only registered accounts.

# 3. System Users (Actors)

The actors will be customers (app users) and company management/staff (labelled as Administrators). The app users are labelled as customers because in the entire purpose of this app is to monetize it in a real world scenario and thus any user when monitized will become a customer.

## 3.1 Actor Descriptions

### 3.1.1 User

* Type:
  + **Guest**: This a public unregistered user who will have limited access such as viewing and searching recipes.
  + **Registered User**: These users will have entire full functionality to the system. This includes adding recipes, sorting by filters, rating recipes, and liking (our own form of favouriting).

### 3.1.2 Corporate

* Type:
  + **Management**: This is company management who wish to use the system for their use and moderation purpose.
  + **Administrators**: These are the administrators who moderate content and handle any user related issues.

# 4. Initial Use Case Identification

## 4.1 ChefsHub Cooking Application System Wide Use Case Diagram

## 4.2 Identify Scenarios

### 4.2.1 SortRecipeByCuisine scenario

|  |  |
| --- | --- |
| *Scenario name* | sort RecipeByCuisine |
| *Participating actor instances* | Bob, User |
| *Flow of events* | 1. Bob, is wondering what he will eat for dinner. He comes to conclusion he wants mediterranean. He opens the cooking application. 2. the cooking application displays dropdown menus . And clicks on mediterranean as a option under the dropdown menu cuisine. 3. All mediterranean cuisine is displayed 4. Bob decides and the information is displayed once Bob picks and option. |

### 4.2.2 DeleteRecipe scenario

*Scenario name:* deleteBowlOfCerealRecipe

*Participating actor instance: Lee*

*Flow of events:*

1. Lee opens the application on his mobile device.

2. Lee clicks the Sign In button and enters his username and password followed by clicking the login button.

4. His login information is verified as correct and he is logged in.

5. Lee clicks the My Profile button and is brought to his profile page.

6. Lee clicks the Delete button beside his Bowl of Cereal recipe.

7. Lee clicks the Yes button when given the confirmation popup.

8. The recipe is removed from both Lee's profile and the database.

### 4.2.3 AddRecipe scenario

*Scenario name:* addBowlOfCornFlakesRecipe

*Participating actor instance: Lee*, User

*Flow of events:*

1. Lee opens the application on his mobile device.

2. Lee clicks the Sign In button and enters his username and password followed by clicking the login button.

3. His login information is verified as correct and he is logged in.

4. Lee clicks the Add Recipe button.

5. Lee enters the Recipe Name: Bowl of Corn Flakes, the Prep and Cook Time: 2 and 0 mins, the Ingredients List: cereal and milk, the Category: Lazy, the Step by Step Directions: pour corn flakes in bowl, pour milk on cereal, enjoy, the Difficulty Rating (out of 5): 1, adds fancy a photo of the bowl of corn flakes, and adds the Number of Servings: 1, Lee does not add any extra notes about the recipe.

6. Lee clicks submit and recipe is added to database along with Lee's username, and the Current Date: October 30, 2014.

### 4.2.4 RateRecipe scenario

*Scenario name* rateCereal

*Participating actor instances Ryan*, User

*Flow of events* 1. Ryan is hungry and wants to find an easy recipe to make, so he opens up the application.

2. Ryan enters his username and password and is logged in to the application.

3. Ryan browses a list of recipes and finds one for a bowl of corn flakes.

4. Ryan clicks on the recipe and is brought to the recipe page. He follows the instructions and makes himself a bowl of cereal.

5. After enjoying his cereal, he looks back at the recipe page and rates the recipe 4 stars.

6. He then closes the application.

### 4.2.5 SortRecipeByPreparationTime scenario

|  |  |
| --- | --- |
| *Scenario name* | sort RecipeByPrepTime |
| *Participating actor instances* | *Bob*, User |
| *Entry Condition*  *Flow of events*  *Exit Condition* | 1. Bob, is wondering what he will eat for dinner. He comes to conclusion he wants something that is fast to make. He opens the cooking application. 2. the cooking application displays dropdown menus . And clicks on Prep Time tab to find fastest meal he can make under the PrepTime tab. 3. Cusinine is sorted by lowest to highest prep time 4. Bob decides and the information is displayed once Bob picks and option. |

### 4.2.6 SortRecipeByDifficulty scenario

|  |  |
| --- | --- |
| *Scenario name* | sort RecipeByDifficulty |
| *Participating actor instances* | Bob, User |
| *Entry Condition*  *Flow of events*  *Exit Condition* | 1. Bob, is wondering what he will eat for dinner. He comes to conclusion he wants something easy to make. He opens the cooking application. 2. the cooking application displays dropdown menus . And clicks on 2 to get recpies to display that are a diffuclty of rating 2 out of 5 . 3. All cuisine rated 2 is displayed 4. Bob decides and the information is displayed once Bob picks and option. |

### 4.2.7 SortRecipeByDate scenario

|  |  |
| --- | --- |
| *Scenario name* | sort RecipeByDate |
| *Participating actor instances* | Bob, User |
| *Entry Condition*  *Flow of events*  *Exit Condition* | 1. Bob, is wondering what he will eat for dinner. He comes to conclusion he wants something new to make. He opens the cooking application. 2. the cooking application displays dropdown menus . And clicks on date tab to display all the recently added recpies. 3. All recently added cuisine is displayed 4. Bob decides and the information is displayed once Bob picks and option. |

### 4.2.8 LikeRecipe scenario

|  |  |
| --- | --- |
| *Scenario name* | LikePizza |
| *Participating actor instances* | Ryan, User |
| *Entry Condition*  *Flow of events*  *Exit Condition* | 1. Ryan is hungry and wants to find an easy recipe to make, so he opens up the application 2. Ryan enters his username and password and is logged in to the application. 3. Ryan browses a list of recipes and finds one for a pizza. 4. Ryan clicks on the recipe and is brought to the recipe page. He follows the instructions and makes himself a pizza. 5. After enjoying his pizza, he looks back at the recipe page and likes the recipe. 6. He then closes the application. |

### 4.2.9 UnlikeRecipe scenario

|  |  |
| --- | --- |
| *Scenario name* | unlikePizza |
| *Participating actor instances* | Ryan, User |
| *Entry Condition*  *Flow of events*  *Exit Condition* | 1. Ryan is hungry and wants to find an easy recipe to make, so he opens up the application 2. Ryan enters his username and password and is logged in to the application. 3. Ryan browses a list of recipes and finds one for a pizza. 4. Ryan clicks on the recipe and is brought to the recipe page. He follows the instructions and makes himself a pizza. 5. After not enjoying his pizza, he looks back at the recipe page and unlike the recipe. 6. He then closes the application. |

## 4.3 Identify Use Cases

### 4.3.1 sortByCuisine use case

|  |  |
| --- | --- |
| *Use case name* | sortByCuisine |
| *Participating actor* | Invoked by User |
| *Entry condition* | 1.The User opens cooking application |
| *Flow of events* | 2. User opens dropdown menu under heading cuisine.  3. User clicks on recipe type which is dislayed as one of the numerous options displayed under cuisine.  4. all cuisine uploaded by users that were identified as selected recipe type are displayed in most recently added order.  5. user clicks on a recipe  6. ingredients ,cooking time , prep time, step by step directions and etc are all displayed under the recipe heading |
| *Exit condition* | 7. User closes app |

### 4.3.2 AddRecipe use case

*Use case name:* addRecipe

*Participating actor instance: User*

Entry Condition: 1. User opens application.

*Flow of events:* 2. User clicks Sign In button and proceeds to enter their username and password in the corresponding fields.

3. User submits login information by clicking the Login button.

4. If login information is verified as correct, user is logged in.

5. User clicks Add Recipe button.

6. User enters Recipe Name, Prep and Cook Time, Ingredients List, Category, Step-by-Step Directions, Difficulty Rating (out of 5), adds a photo, Number of Servings, Extra Notes (optional).

Exit Condition: 7. User clicks submit and recipe is added to database along with Username of user, Current Date.

### 4.3.3 DeleteRecipe use case

*Use case name:* deleteRecipe

*Participating actor instance: User*

Entry condition: 1. User opens application.

*Flow of events:* 2. User clicks Sign In button and proceeds to enter their username and password in the corresponding fields.

3. User submits login information by clicking the Login button.

4. If login information is verified as correct, user is logged in.

5. User clicks the My Profile button and is taken to their profile page.

6. User scrolls through their list of recipes and beside the desired recipe, clicks the Delete Recipe button.

7. A confirmation popup is displayed with text "Are you sure you would like to delete this recipe?" along with a Yes button and a No button.

Exit Condition: 8. If User clicks yes, the recipe is removed from both their profile and the database.

### 4.3.4 RateRecipe use case

*Use case name*  The RateRecipe

*Participating actor* Invoked by User

*Entry condition* 1. User opens application

*Flow of events* 2. User clicks Sign In button and proceeds to enter their username and password in the corresponding fields.  
3. User submits login information by clicking the Login button.  
4. If login information is verified as correct, User is logged in.

5. User browses the database of recipes and selects one.

6. The application brings up the recipe page including all details about the recipe and a five star rating system where a sequence of five star images is shown.

7. User clicks on the number of stars they wish to rate the recipe as (out of 5).

8. The database is updated by setting the recipe’s rating to the average number of stars it received from all of its ratings.

*Exit condition* 9. User closes the application.

### 4.3.5 SortByPreparationTime use case

|  |  |
| --- | --- |
| *Use case name* | sortByPrepTime |
| *Participating actor* | Invoked by User |
| *Entry condition* | 1.The User opens cooking application |
| *Flow of events* | 2. User clicks Sign in button and proceeds to enter their username and password in the corresponding fields.  3. User submits login information by clicking the Login button.  4. If login information is verified as correct, user is logged in.  5. User opens dropdown menu under heading Prep Time.  6. User clicks on lowest to highest which is dislayed as one of two options displayed under PrepTime.  7. all cuisine uploaded by users that were identified as minimal preptime are displayed in order.  8. User clicks on Macaroni and cheese  9. ingredients, cooking time, prep time, step by step directions and etc are all.  displayed under the Macraoni and cheese. |
| *Exit condition* | 10. User finishes cooking and closes app. |
| 4.3.6 SortByDifficulty use case |  |

|  |  |
| --- | --- |
| *Use case name* | sortByDifficulty |
| *Participating actor* | Invoked by User |
| *Entry condition* | 1.The User opens cooking application |
| *Flow of events* | 2. User clicks Sign in button and precedes to enter their username and password in the corresponding fields.  3.user submits login information by clicking the Login button.  4.If login information is verified as correct, user is logged in.  5. User opens dropdown menu under heading Difficulty.  6. User clicks on 2 which is dislayed as one of the 5 options displayed under Difficulty.  7. all cuisine uploaded by users that were identified as a rating difficulty of 2 are displayed in most recently added order.  8. user clicks on Cereal.  9. ingredients ,cooking time , prep time, step by step directions and etc are all displayed under the Cereal |
| *Exit condition* | 10. User finishes cooking and closes app |

### 4.3.7 SortByDate use case

|  |  |
| --- | --- |
| *Use case name* | sortByDate |
| *Participating actor* | Invoked by User |
| *Entry condition* | 1.The User opens cooking application. |
| *Flow of events* | 2. User clicks Sign in button and precedes to enter their username and password in the corresponding fields.  3.user submits login information by clicking the Login button.  4.If login information is verified as correct, user is logged in.  5. User opens dropdown menu under heading cuisine.  6. User clicks on recent which is dislayed as one of the few options displayed under cuisine.  7. all cuisine uploaded by users that were uploade within the last week are displayed in most recently added order.  8. user clicks on pizza.  9. ingredients ,cooking time , prep time, step by step directions and etc are all displayed under the pizza. |
| *Exit condition* | 10. User finishes cooking and closes app. |
| 4.3.8 LikeRecipe use case |  |

*Use case name*  The LikeRecipe

*Participating actor* Invoked by User

*Entry condition* 1. User opens application

*Flow of events* 2. User clicks Sign In button and proceeds to enter their username and password in the corresponding fields.  
3. User submits login information by clicking the Login button.  
4. If login information is verified as correct, User is logged in.

5. User browses the database of recipes and selects one.

6. The application brings up the recipe page including all details about the recipe and a like/dislike system where a like/dislike image is shown.

7. User clicks on the like if they wish to bookmark it in their profile.

8. The database for the user is updated by adding the recipes to its liked database.

*Exit condition* 9. User closes the application.

### 4.3.9 UnlikeRecipe use case

*Use case name*  The UnlikeRecipe

*Participating actor* Invoked by User

*Entry condition* 1. User opens application

*Flow of events* 2. User clicks Sign In button and proceeds to enter their username and password in the corresponding fields.  
3. User submits login information by clicking the Login button.  
4. If login information is verified as correct, User is logged in.

5. User browses the database of recipes he’s liked in the past.

6. User clicks on the unlike to remove the bookmark to the recipe.

8. The database for the user is updated by removing the recipes to its liked database.

*Exit condition* 9. User closes the application.

# 5. Non-Functional Requirements

The team has identified the following non-functional requirements for this project:

* The system must be scalable and expandable as this is one the major things an online application should look at considering the future dynamics of this application might take a turn requiring the system to be scaled for drastically high amount of users.
* The system must handle a high amount of concurrent users.
* The final frontend system must be easy to update and maintain.
* The backend system must be return responses to the frontend in less than 10 seconds. 10 seconds being the maximum allowed time for any API request call.
  + This will in turn make our frontend seem “faster” to the general user as the application will seem fast, snappy, and responsive.
* The application frontend (UI) must be straightforward and should promote ease of use functionality. The general non-tech savvy user without any technical knowledge must be able to use this with no difficulty.

**ChefsHub: A recipe sharing platform for culinary enthusiasts.**

**Analysis Document**

Dr. Chính Hoàng

Thursday, Nov. 27, 2014

Group #4 (Ryan Burke, Ranadeep Polavarapu, Lee Glendenning, Bruno Salapic)

****

Table of Contents

1. Initial Object Identification from Use Cases 21

1.1 Entity, Boundary, Control Objects Description 21

1.2 Initial Objects from Use Cases 21

1.2.1 Initial Objects from SortByCuisine use case 21

1.2.1 Initial Objects from AddRecipe use case 21

1.2.2 Initial Objects from DeleteRecipe use case 22

2. CRD Cards for Initial Objects 22

2.1 CRC Cards Overview 22

2.2 CRC Cards (Class Responsibility Collaboration) 22

2.2.1 CRC Card for class User 22

2.2.2 CRC Card for class MainController 23

2.2.3 CRC Card for class LoginController 23

2.2.4 CRC Card for class Recipe 24

3. Operations and Attributes of Objects 24

4. State Diagrams 25

4.1 State Diagram Overview 25

4.2 Recipe State Diagram 25

4.3 User State Diagram 26

5. Sequence Diagrams 26

5.1 Sequence Diagram Overview 26

5.2 Rate Recipe Sequence Diagram 27

5.3 DeleteRecipe Sequence Diagram 28

5.4 AddRecipe Sequence Diagram 29

5.5 SortByCuisine Sequence Diagram 30

5.6 LikeRecipe Sequence Diagram 30

5.7 SortByDate Sequence Diagram 31

5.8 SortByDifficulty Sequence Diagram 31

5.9 UnlikeRecipe Sequence Diagram 32

Analysis

# 1. Initial Object Identification from Use Cases

## 1.1 Entity, Boundary, Control Objects Description

The objects identified for the use case of rating a recipe. **Entity objects** represent the persistent information tracked by the system. **Boundary Objects** represent the system interface with the actors. In each use case, each actor interacts with at least one boundary object. **Control Objects** are responsible for the coordination of boundary and entity objects.

## 1.2 Initial Objects from Use Cases

### 1.2.1 Initial Objects from SortByCuisine use case

The identified objects for **SortByCuisine** use case.

|  |  |  |
| --- | --- | --- |
| **Entity** | **Boundary** | **Control** |
| User   * Username * Password   Recipe | App start button  Device screen   * Home screen * Search results screen * Recipe screen   Device home button | Main control |

### 1.2.1 Initial Objects from AddRecipe use case

Identifying all the objects for the **AddRecipe** use case.

|  |  |  |
| --- | --- | --- |
| **Entity** | **Boundary** | **Control** |
| User   * username * password * recipe (all details required to create the recipe) | App start button  Device screen   * Home screen * Login screen * Search results screen * Recipe screen * MyProfile button * User profile screen * Add recipe button * Submit recipe button   Device home button | Main control  Login control |

## 1.2.2 Initial Objects from DeleteRecipe use case

Identifying all the objects for the **DeleteRecipe** use case.

|  |  |  |
| --- | --- | --- |
| **Entity objects** | **Bounday Objects** | **Control Objects** |
| User   * Username * Password * Recipe | Device screen   * Main page * Login page * Recipe page * Search result page * My Profile page * My Profile button * Delete Recipe button * Confirmation popup message * Yes and no buttons confirming the deletion of a recipe   Device home button | Main Control  Login Control |

# 2. CRD Cards for Initial Objects

## 2.1 CRC Cards Overview

Objects are described by class-responsibility-collaboration (CRC) cards. A CRC card contains the name of the class, the functionality (responsibilities) of the class and a list of the other class it depends on in order to fulfill its responsibilities (collaboration).

## 2.2 CRC Cards (Class Responsibility Collaboration)

### 2.2.1 CRC Card for class User

|  |
| --- |
| **Class**  User |
| **Responsibility**   1. Store username 2. Store password 3. Store recipes added by user 4. Store user profile information 5. Store recipes liked 6. Store user status (active or banned) |
| **Collaboration**   1. Class **MainController** 2. Class **LoginController** |

|  |
| --- |
| **Class**  MainController |
| **Responsibility**   1. Display pages 2. Queries user database 3. Queries recipe database 4. Creates login controller |
| **Collaboration**   1. Class **LoginController** 2. Class **Recipe** |

### 2.2.2 CRC Card for class MainController

### 2.2.3 CRC Card for class LoginController

|  |
| --- |
| **Class**  LoginController |
| **Responsibility**   1. Queries user database 2. Display login page 3. Create new account 4. Add to user database |
| **Collaboration**   1. Class **User** 2. Class **MainController** |

### 2.2.4 CRC Card for class Recipe

|  |
| --- |
| **Class**  Recipe |
| **Responsibility**   1. Store recipe name 2. Store prep and cook time 3. Store ingredients list 4. Store cuisine category 5. Store step-by-step directions 6. Store difficulty rating 7. Store recipe photo 8. Store number of servings 9. Store extra notes 10. Query recipe database 11. Add to recipe database |
| **Collaboration**   1. Class **MainController** 2. Class **User** |

# 3. Operations and Attributes of Objects

User Object

Attributes

* Username
* Password
* Account status

Operations

* Create user account
* Delete user account
* Ban user account
* Like recipe
* Unlike recipe

Recipe Object

Attributes

* Recipe name
* Prep and cook time
* Ingredients list
* Cuisine category
* Step-by-step directions
* Difficulty rating
* Recipe photo
* Number of servings
* Extra notes
* Rating

Operations

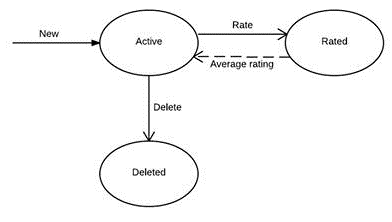
* Add
* Delete
* Rate

# 4. State Diagrams

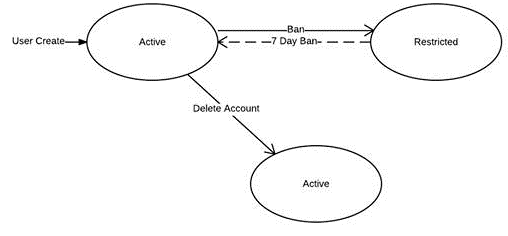
## 4.1 State Diagram Overview

State Chart Diagrams describe the behavior of an individual object as a number of states and transitions between these states.

## 4.2 Recipe State Diagram



## 4.3 User State Diagram

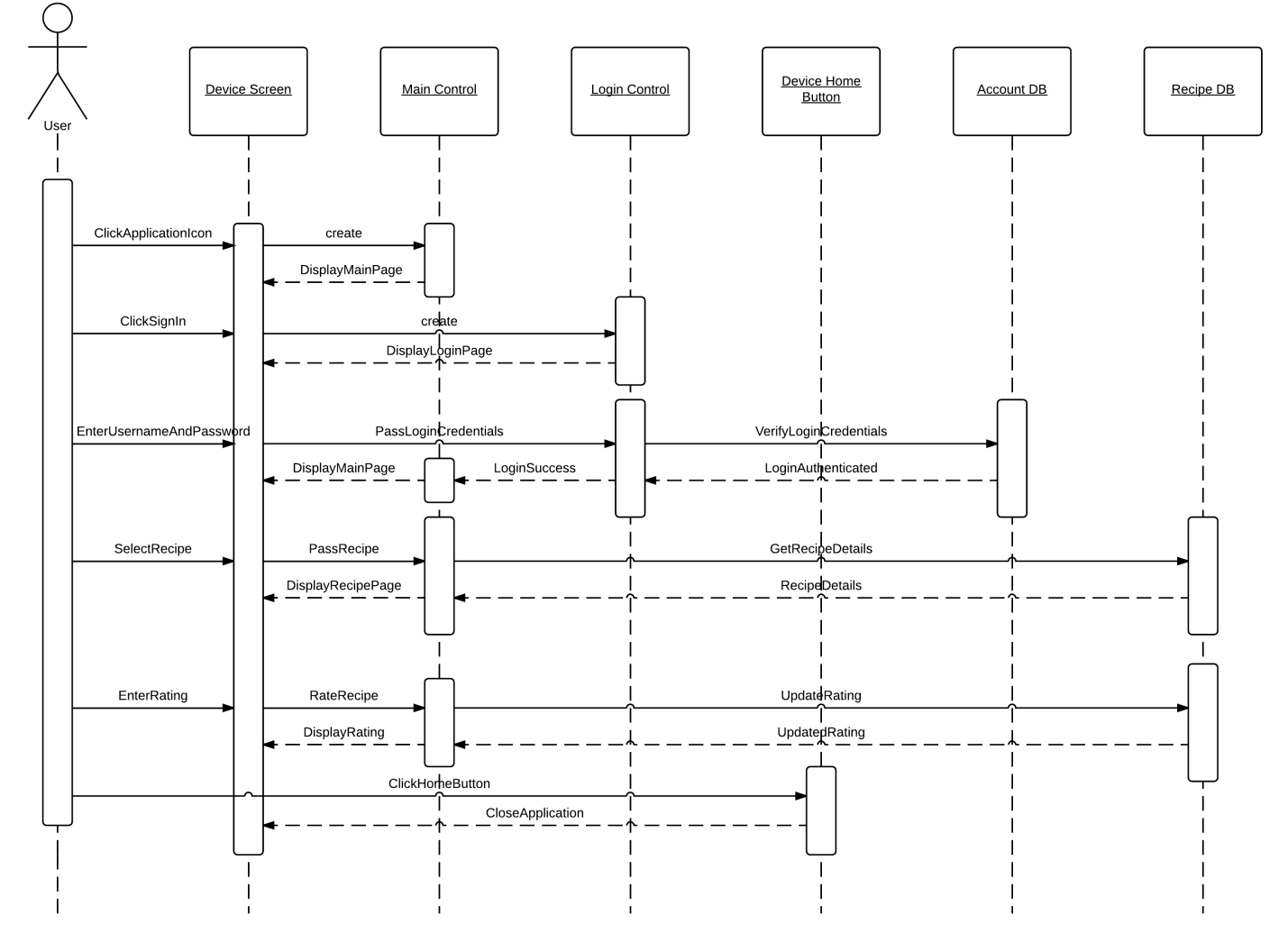


# 5. Sequence Diagrams

## 5.1 Sequence Diagram Overview

Sequence diagrams are used to formalize the behavior of the system and to visualize the communication among objects. Sequence diagrams must be associated with a use case. They contain the objects (entity, boundary, control) of that use case.

## 5.2 Rate Recipe Sequence Diagram



## 5.3 DeleteRecipe Sequence DiagramC:\Users\Lee\Desktop\Analysis\Analysis - Delete Recipe Sequence Diagram.png

## 5.4 AddRecipe Sequence DiagramC:\Users\Lee\Desktop\Analysis\ihatehoang - Page 1.png

## 5.5 SortByCuisine Sequence DiagramC:\Users\Lee\Desktop\Analysis\Like Recipe Sequence Diagram - New Page.png

## 5.6 LikeRecipe Sequence DiagramC:\Users\Lee\Desktop\Analysis\Like Recipe Sequence Diagram.png

## 5.7 SortByDate Sequence DiagramC:\Users\Lee\Desktop\Analysis\Sort by Date Posted Sequence Diagram.png

## 5.8 SortByDifficulty Sequence DiagramC:\Users\Lee\Desktop\Analysis\Sort by Difficulty Sequence Diagram .png

## 5.9 UnlikeRecipe Sequence DiagramC:\Users\Lee\Desktop\Analysis\Unlike Recipe Sequence Diagram.png

**ChefsHub: A recipe sharing platform for culinary enthusiasts.**

**Design Document**

Dr. Chính Hoàng

Thursday, Nov. 27, 2014

Group #4 (Ryan Burke, Ranadeep Polavarapu, Lee Glendenning, Bruno Salapic)

****

Table of Contents

1. System Design 36

1.1 Subsystem Design 36

1.2 High-Level Subsystem Design 37

1.3 Subsystem Responsibilities 37

1.4 Dependencies 38

1.5 Subsystem to Hardware Mapping 39

1.5.1 Subsystem to Hardware Mapping Overview 39

1.6 Access Control 40

1.7 Data Storage 40

1.8 Design Rationale 40

2. Object Design 41

2.1 Entity Objects 41

2.2 Boundary Objects 41

2.3 Control Objects 45

2.4 Class Diagram 45

2.4.1 User class 45

2.4.2 Recipe class 45

2.4.3 LoginController class 46

2.4.4 MainController class 47

2.5 Cohesion & Coupling Analysis Between Subsystems 48

2.5.1 Cohesion 48

2.5.2 Coupling 48

3. Data Flow Diagrams of Subsystems 49

3.1 Gane and Sarsen’s Analysis (Data Flow Diagram [DFD] based on Gane and Sarsen’s rules) [High-level DFD] 49

3.2 Subsystem specific Data Flow Diagrams 50

3.2.1 RateRecipe DFD 50

3.2.2 FilterRecipes DFD 50

3.2.3 RecipeManagement DFD 51

3.2.4 ManagementInterface DFD 51

3.2.5 Like/DislikeRecipe DFD 52

3.2.6 UserManagement DFD 52

Design

# 1. System Design

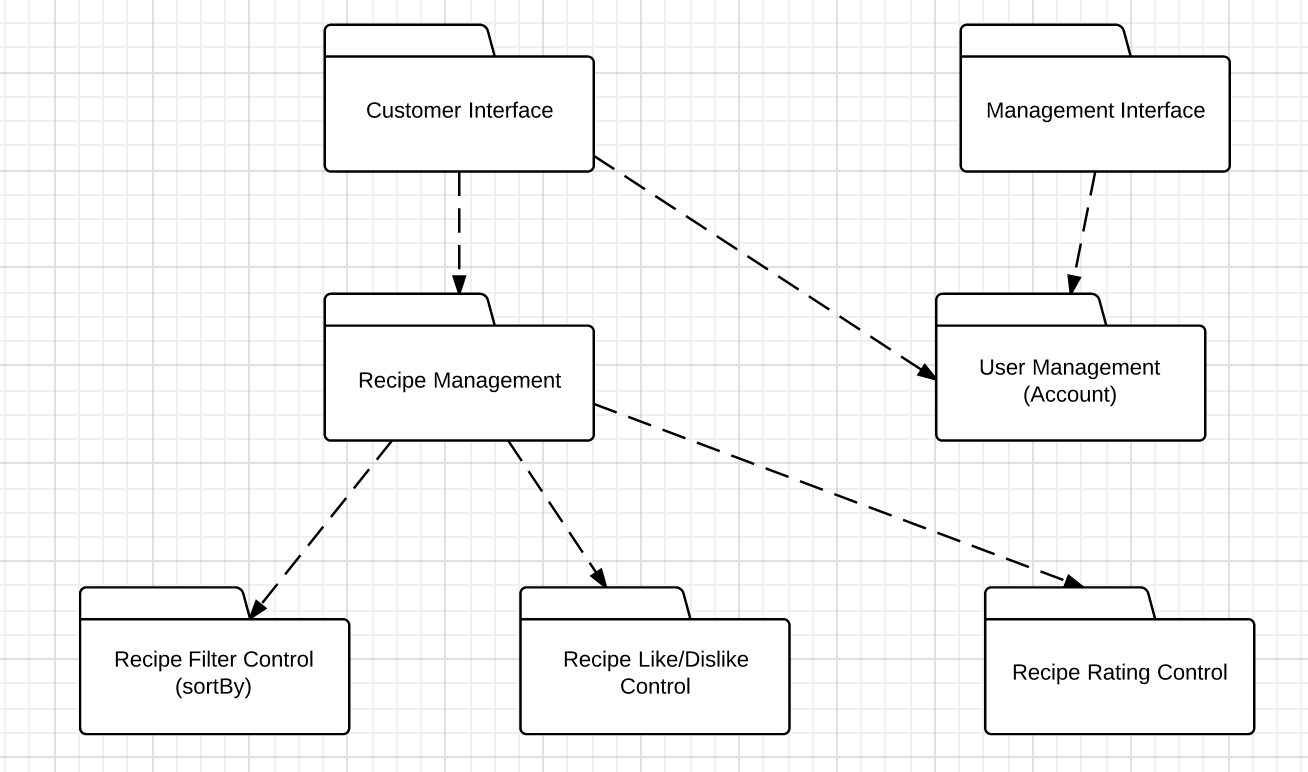
## 1.1 Subsystem Design

* Our system will be composed of seven (7) major subsystems being:

1. **Customer Interface**
2. **Management Interface (Administrator and Corporate)**
3. **User Management**
4. **Recipe Management**
5. **Recipe Filter Control (sortBy)**
6. **Recipe Like/Dislike Control**
7. **Recipe Rating Control**

* The approach to designing the seven major subsystems was to break down the subsystems to the lowest level where one subsystem could be implemented by one developer.

## 1.2 High-Level Subsystem Design



## 1.3 Subsystem Responsibilities

1. **Customer Interface**: This subsystem is responsible for the handling all UI design and user interface aspect of the application. Meaning that this interface will complete the link of human-machine interaction (machine being Android or some other mobile device) and will only serve the application’s users. This subsystem will have only features available to users.
2. **Management Interface**: This is the UI and user interface for management and corporate in a real world scenario. This interface provides extended management features such as banning abusive users and deleting recipes that are deemed offensive and abusive in terms of content.
3. **User Management**: This subsystem is responsible for all account related functionality such as editing profile and deactivating your account. This subsystem was also made to be responsible for working parallel with payment management subsystem should that need arise in the future where such a cooking app provides exclusive features who purchase an account subscription.
4. **Recipe Management**: This subsystem is responsible for handling all recipe related functionality such as creating recipes, editing an existing recipe, and deleting a recipe. Other functionality such as rating, like/dislike, and sorting are siphoned off to the subsystems below, as they are major features that require their own subsystems to keep this current one uncluttered.
5. **Recipe Filter Control**: This subsystem is responsible for returning filter/sort by data to the recipe management subsystem based on certain parameters that are predetermined. These parameters being like sort by upload date, popularity, rating, etc.
6. **Recipe Like/Dislike Control**: This subsystem is responsible for handling all the like and dislike related functionality.
7. **Recipe Rating Control**: This subsystem is responsible for handling the rating of a recipe (ex. Rate this recipe 5/5 stars) functionality.

## 1.4 Dependencies

* The dependencies are outlined by the **dotted lines** in the high-level subsystem design diagram. The most crucial dependencies in our system design are between **Recipe Management** and **Recipe Filter Control, Recipe Like/Dislike Control, Recipe Rating Control**. The **Recipe Management** subsystem depends on those three subsystems to ensure it has full functionality in the handling of recipes.

## 1.5 Subsystem to Hardware Mapping

## 1.5.1 Subsystem to Hardware Mapping Overview

The diagram depicted above shows the subsystems as layered and mapped to the corresponding **middleware**, **software**, and **hardware**. As per specific subsystem mapping to the hardware the Android hardware will be using TCP/IP networking protocol will be communicating directly with the web browser which will render the front end interface and it will be communicating with our backend (which handles the application specific subsystems) to perform functions such as adding recipes, user logins, and etc.

## 1.6 Access Control

The access control is limited between each subsystem. In specific to the *Application Specific Layer* the there is access between *Recipe Management* and the *Recipe Filter Control*, *Recipe Like/Dislike Control*, and *Recipe Rating Control*. There is also access between the *Recipe Management* and *User Management* however; the access is restricted in this partition level. There is only read access in this partition level between subsystems. The write and execute access for data is restricted to each particular subsystem itself.

## 1.7 Data Storage

Data storage for the *Application Specific Layer* will be done through the *Django* backend. *Django* is a Python MVC web framework that will handle all data communicate and data modification via a RESTful API. *Django* will create a link between itself and *PostgreSQL*. *PostgreSQL* is an object-relational database management system (ORDBMS) will manage and store the data. *Django* will also handle certain *Application Specific Layer* data that will be queried very frequently so some sort of caching is required to ensure API call response times are less than **1.5s**. This data will be handled and stored in an in-memory key value store like *Redis* or *memcached* to ensure high-speed data retrieval.

## 1.8 Design Rationale

We as a group agreed on a system design, which is a combination of both layering and partitioning.

We chose layering because we needed hierarchical structure from layering to define parent child relationship for certain classes to avoid class cluttering and over complication. We follow Object Oriented Programming (OOP) so we definitely needed the hierarchical structure. We have an open architecture meaning that some layers will access layers at deeper levels (ex. Django backend [Middleware layer] accesses *Recipe Management* and *User Management* [Application Specific Layer]).

As per partitioning, we chose this because we needed some subsystems to work directly and communicate with other subsystems in that *partition*. For ex. *Recipe Management* subsystem communicates with *User Management* in parallel to determine whether someone has the sufficient permissions to post a recipe or edit a recipe (we don’t want someone editing a recipe that doesn’t belong to them). Thus, the following reasons why we went with a combination of both layering and partitioning when choosing a system design.

# 2. Object Design

**Object Descriptions**

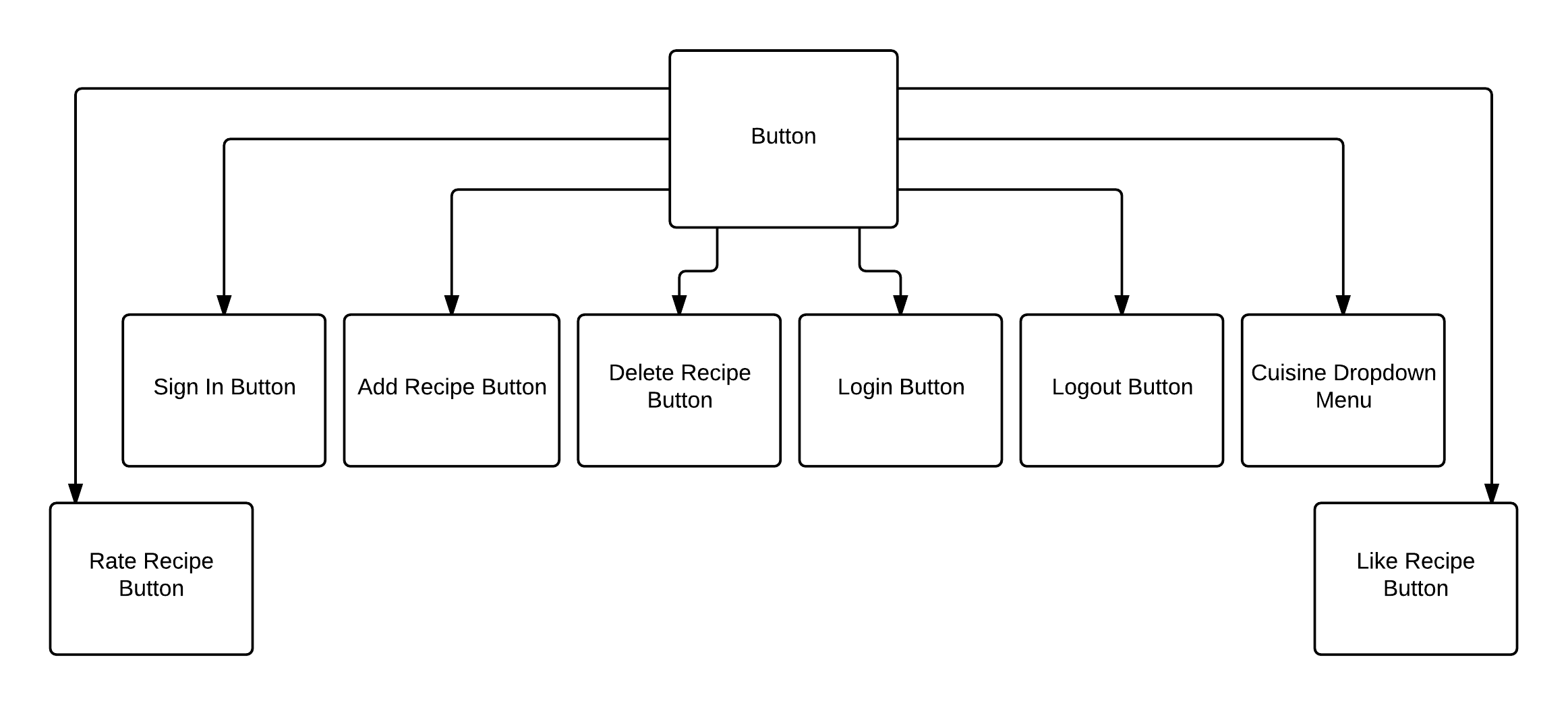
## 2.1 Entity Objects

* **User**
  + This object stores both a username and password for login validation and profile information such as recipes added by user, personal profile information, recipes liked and user status (active or banned). The login control object and the main control object use this object.
* **Recipe**
  + This object stores information regarding a recipe such as the recipe name, the prep and cook time, the ingredients list, the cuisine category, step by step directions, the difficulty rating, a photo of the dish, the number of servings, the username of the user who posted the recipe, and any extra notes. The main controller object to add or pull query results from the recipe database uses this object.

## 2.2 Boundary Objects

**Button Classes**

Button classes are used in the application to notify the main controller of a user’s intent to perform an action. Each button object is derived from the abstract class Button.



Sign In Button

* An instance of this button is contained by the Main Page object. When clicked, the Main Controller object displays the Login Page object and creates an instance of a Login Controller object.

Login Button

* An instance of this button is contained by the Login Page Object. When clicked, the Login Controller object queries the User Database to verify the inputted user information.

Cuisine Dropdown Menu

* An instance of this button is contained by the Main Page Object. When a specific cuisine button is clicked, the Main Controller object queries the database for corresponding recipe instances.

Like Recipe Button

* An instance of this button is contained by Recipe Page Objects. When clicked, the Main Controller object makes corresponding changes to the User in the User Database.

Rate Recipe Button

* An instance of this button is contained by Recipe Page Objects. When clicked, the Main Controller object makes corresponding changes to the Recipe in the Recipe Database.

Add Recipe Button

* An instance of this button is contained by User Profile Page Object. When clicked, the Main Controller object displays an Add New Recipe Form.

Delete Recipe Button

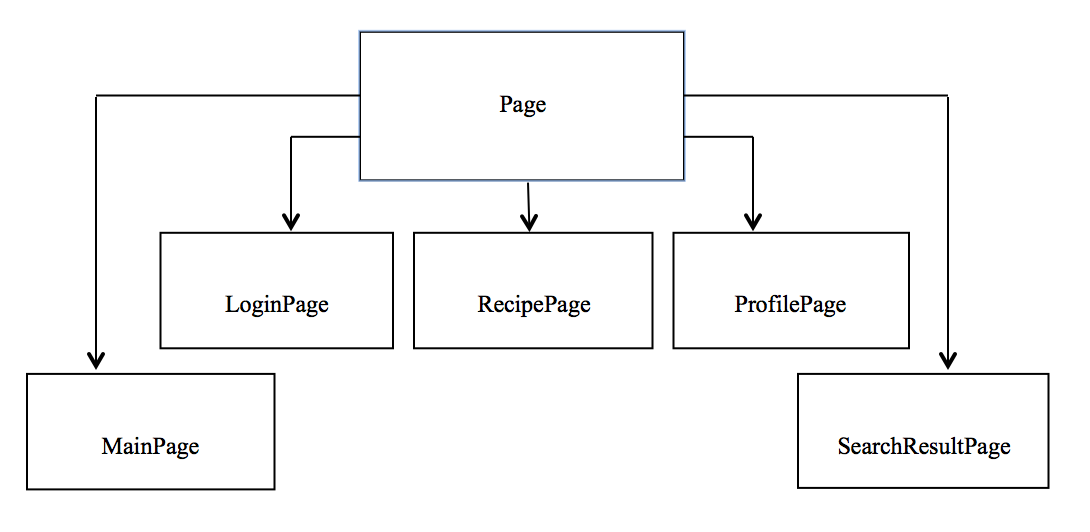
* Instances of this button are contained by the User Profile Page Object. When clicked, the Main Controller object deletes the corresponding recipe from the Recipe Database and makes necessary changes to the User Database

Logout Button

* An instance of this button is contained in every instance of a Page. When clicked, the Main Controller object logs out the user.

Page Classes

* Page classes are used in the application whenever a user navigates to the corresponding page. Each page consists of information regarding the other objects used on the page and various buttons that perform functions or link to other pages. Each page class is a child derived from the abstract class, Page.



MainPage Class

* This is the main page the user sees when opening the application. This page accesses the recipe database and displays a few recently added recipes. Clicking on any of these will create an instance of the RecipePage class for that recipe. This page also contains an instance of the sign in button. When clicked, the main controller creates an instance of the LoginPage class.

LoginPage Class

* This page contains input fields for the user to enter their username and password. It also contains an instance of the login button, which validates the user’s username and password and logs them into the system.

RecipePage Class

* This is the page that is displayed when a recipe is clicked from the main page, the search result page or the profile page. This page contains information about the recipe such as rating, ingredients, cuisine category, etc. This page also contains an instance of the rating button, which allows users to rate the recipe.

ProfilePage Class

* This page contains user information such as name, recipes liked, etc. It also contains a list of recipes added by the user. This page has an instance of the add recipe button and an instance of the delete recipe button so the user can perform those actions.

SearchResultPage Class

* This page displays the list of recipes in the order determined by the search that was performed. They are arranged by date, preparation time, cuisine type or difficulty. When a recipe is clicked, an instance of the RecipePage class is created for that recipe.

Form Class

* Form classes are used when a user is required to enter input. There are a variety of input fields followed by a button to submit the information. Each form class is a child derived from the abstract class, Form.

Form

AddNewRecipeForm

LoginForm

AddNewRecipeForm Class

* This form allows users to enter information pertaining to the recipe they are adding to the database. This form has an instance of the add recipe button which stores the information in the recipe database.

LoginForm Class

* This form allows users to enter their username and password so they can log in to the system. This form has an instance of the login button, which accesses the user database and validates the user’s login information before logging them into the system.

## 2.3 Control Objects

* Main Control
  + The main controller is used whenever the user wants to interact with the app, depending on what the user taps or inputs on the screen it is all sent to the main controller where it is processed and necessary actions are performed such as interacting with the recipe database or changing pages.
* Login Control
  + The login controller is used when the user wants to login to his account. When the user inputs his username and password it would access the database to verify if he has a legitimate account that isn’t banned or if the account is an active account. The control is then passed to the main controller object

## 2.4 Class Diagram

### 2.4.1 User class

|  |
| --- |
| User class |
| Username : String  Password : String  Recipe : objects  profileInfo :Array  recipeLiked :Linked List  UserStatus: Boolean |
| Public User(all above attributes ); |

**Method Descriptions**

Public user(all attributes)

* Access user database
* Add new entry to user database using attributes passed

### 2.4.2 Recipe class

|  |
| --- |
| Recipe class |
| recipeName: string  prepCookTime : integer  IngredientList : list  cusinieCategory: String  step-by-step :String  dateIn : String  rating : Integer  photo :Picturefile  servings : Integer  extraNotes : String |
| Public Recipe(all above attributes ); |

**Method Descriptions**

Public recipe(all attributes)

* Access recipe Database
* Add new entry to recipe database using attributes passed

### 2.4.3 LoginController class

|  |
| --- |
| LoginController class |
| *No Variables for Controller classes.* |
| Public DisplayProfile(array profileInfo);  Private VerifyUser(Username,password); |

**Method Descriptions**

Public DisplayProfile(array profileInfo)

* Create instance of ProfilePage object using profile info to fill out page

Public VerifyUser(string username, string password)

* Access user database
* Search database using username and password
* If username and password match an existing account
* Log user in
* Else
* Login failed

### 2.4.4 MainController class

|  |
| --- |
| Main Controller class |
| *No Variables for Controller classes.* |
| Public Main();  Public AddRecipe(string recipeName, double prepCookTime, string ingredientList, string cusinineCategory, string step-by-step, string dateIn, int rating, string photo, int servings, string extraNotes);  Public DeleteRecipe();  Public SortByCusinine(string cusinieCategory);  Public SortByDifficulty(int rating);  Public SortByDate(string dateIn);  Public SortByPrepTime(double prepCookTime);  Public UserStatus(string user);  Public CreateLoginControl(string username, string password); |

**Method Descriptions**

Public Main()

* Create the main controller class

Public AddRecipe(string recipeName, double prepCookTime, string ingredientList, string cuisineCategory, string stepByStep, string dateIn, int rating, string photo, int servings, string extraNotes)

* Access recipe database
* Add new entry using attributes passed

Public DeleteRecipe()

* Access recipe database
* Remove entry for recipe to be deleted

Public SortByCuisine(string cuisineCategory)

* Access recipe database
* Find all recipes with matching cuisine category
* Display results

Public SortByDifficulty(int rating)

* Access recipe database
* Find all recipes with matching difficulty
* Display results

Public SortByPrepTime(double prepCookTime)

* Access recipe database
* Find all recipes with matching prep time
* Display results

Public UserStatus(string user)

* Access user database
* Determine user status for selected user (ex. banned, active)

Public CreateLoginControl(string username, string password)

* Create instance of login control class

## 2.5 Cohesion & Coupling Analysis Between Subsystems

### 2.5.1 Cohesion

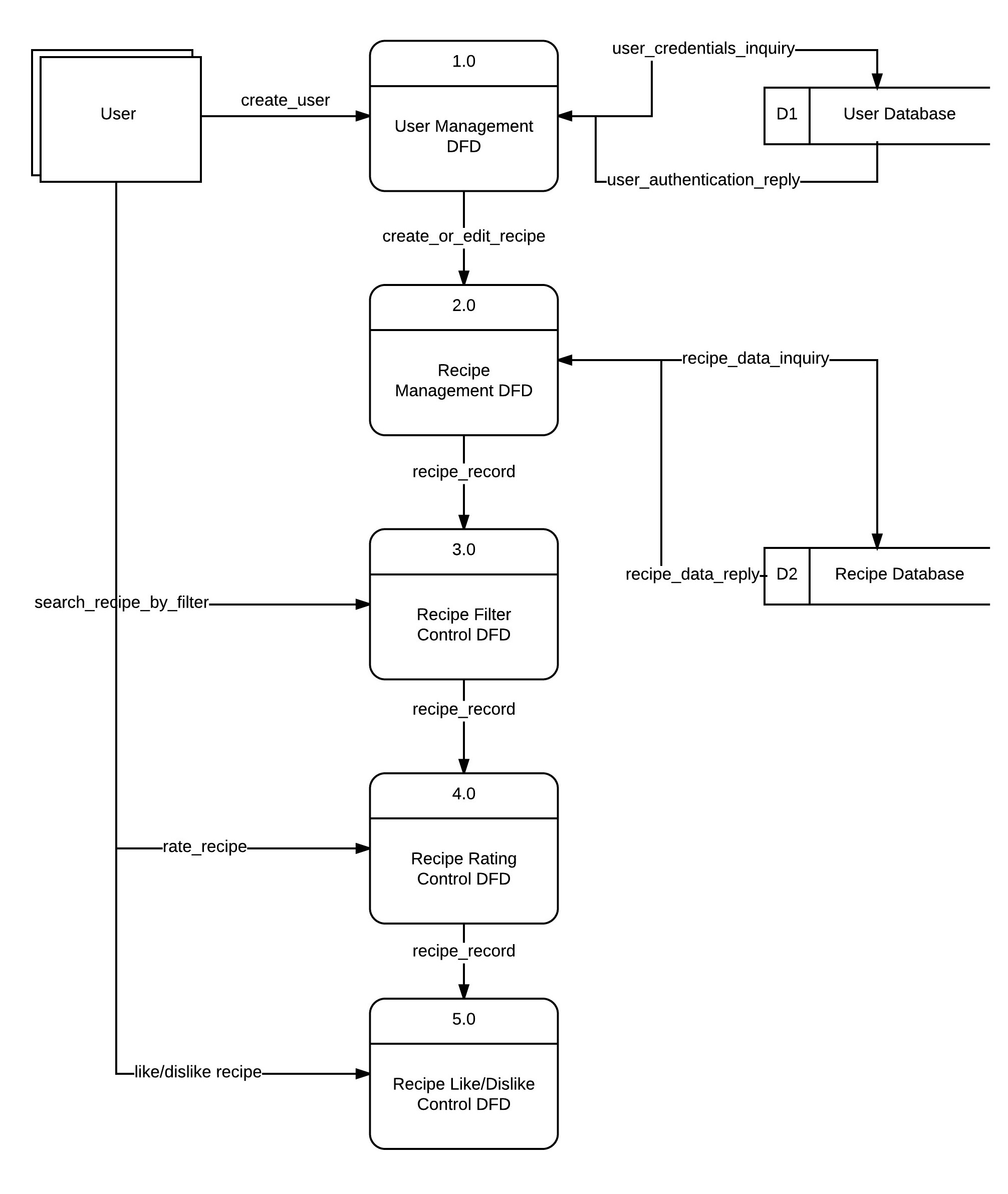
* All of our methods (modules) have either Information Cohesion or Functional Cohesion because they either perform a single action/change on/to a single data structure or perform a number of actions/changes on/to a single data structure. Because of this, our methods are easy to reuse and debug.

### 2.5.2 Coupling

* All of our methods (modules) have either Stamp Coupling or Data Coupling because they either operate on most or all of the elements of the passed data structure.

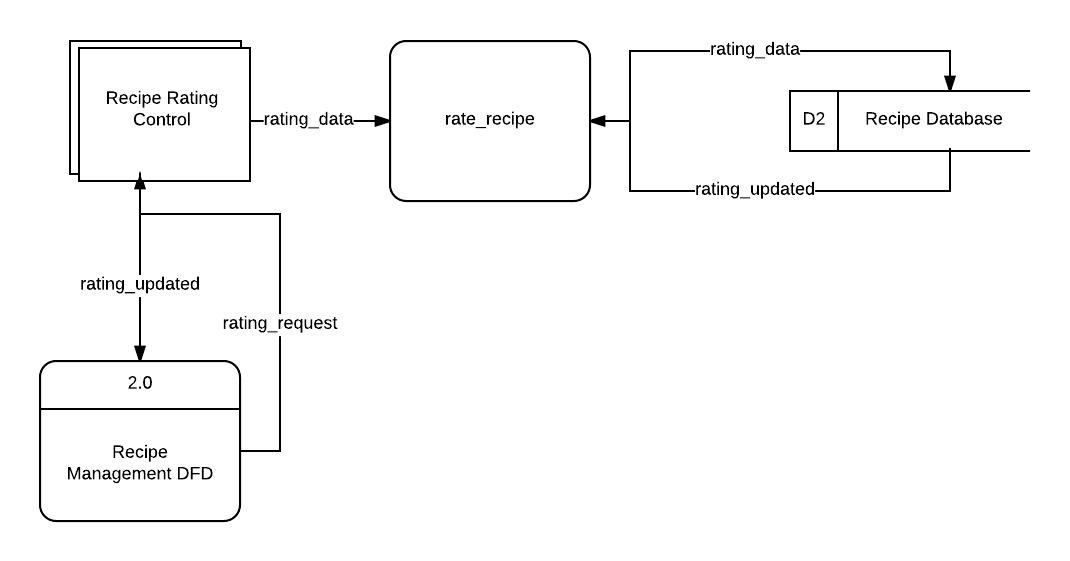
# 3. Data Flow Diagrams of Subsystems

## 3.1 Gane and Sarsen’s Analysis (Data Flow Diagram [DFD] based on Gane and Sarsen’s rules) [High-level DFD]



## 3.2 Subsystem specific Data Flow Diagrams

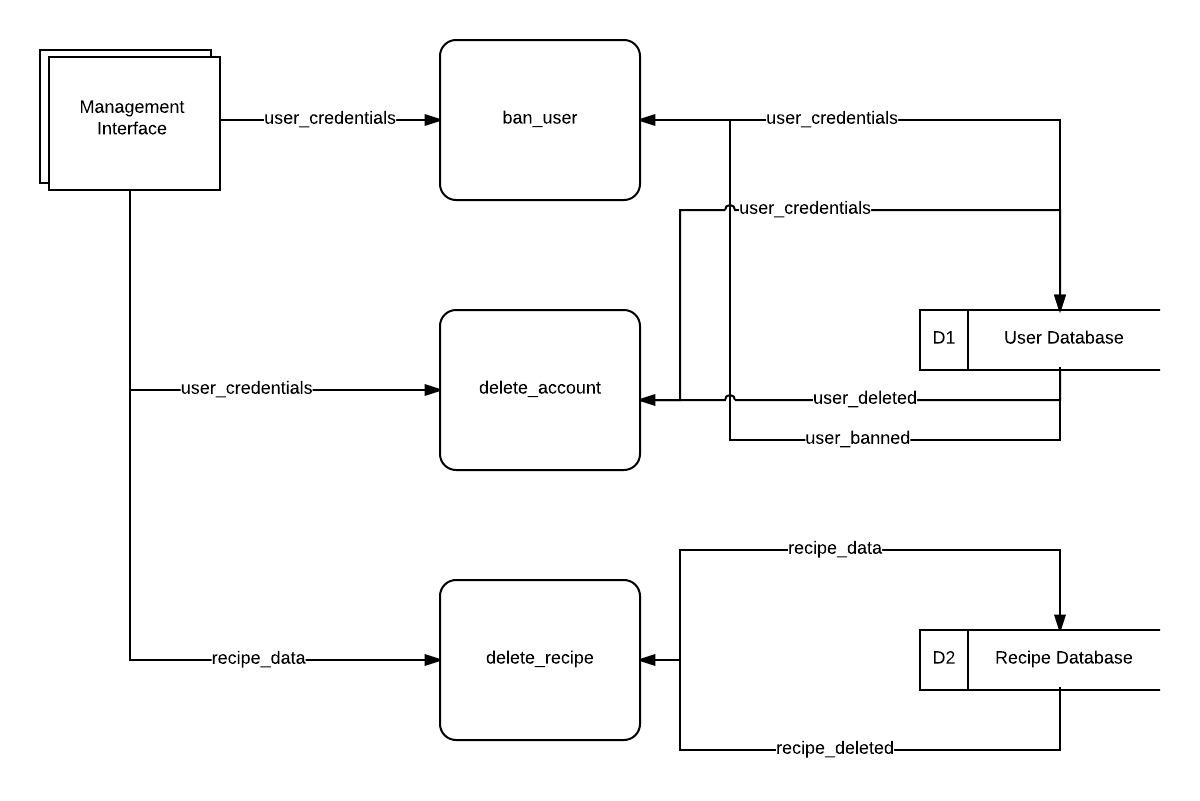
### 3.2.1 RateRecipe DFD



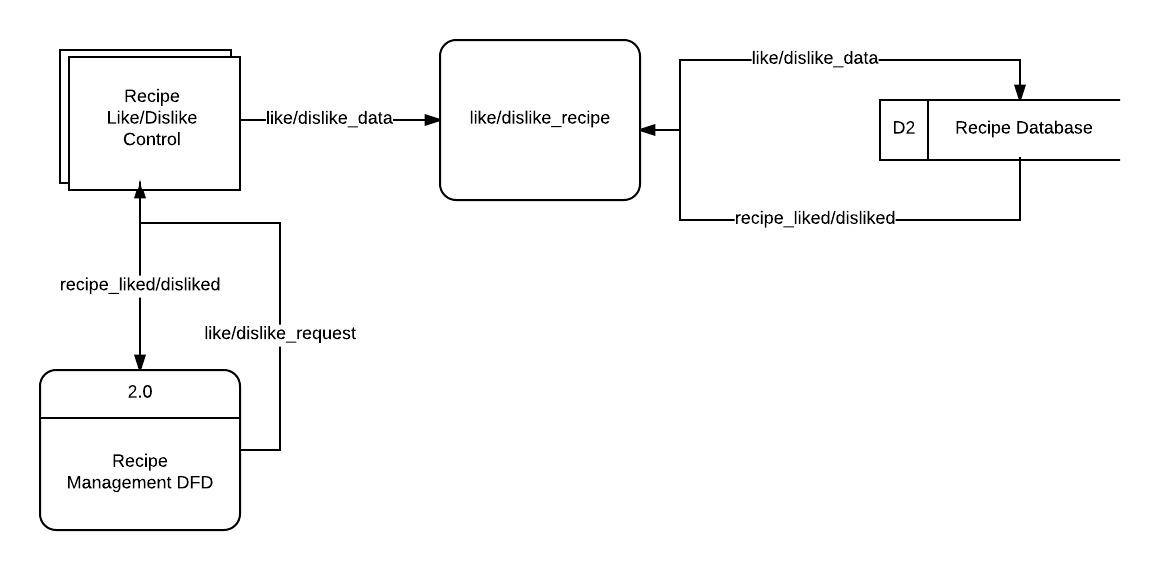
### 3.2.2 FilterRecipes DFDC:\Users\Lee\Desktop\Design\DFDs\Recipe Filter Control - New Page(1).jpeg

### 3.2.3 RecipeManagement DFDC:\Users\Lee\Desktop\Design\DFDs\Recipe - New Page.jpeg

### 3.2.4 ManagementInterface DFD



### 3.2.5 Like/DislikeRecipe DFD



### 3.2.6 UserManagement DFDC:\Users\Lee\Desktop\Design\DFDs\User - New Page(1).jpeg

1. Section 1: Food in Canada. (n.d.). Retrieved November 27, 2014, from http://www.statcan.gc.ca/pub/16-201-x/2009000/part-partie1-eng.htm [↑](#footnote-ref-1)
2. Chart 1.1Personal spending in Canada on food and alcohol as a percent of total personal spending, 1961 to 2008. (n.d.). Retrieved November 27, 2014, from http://www.statcan.gc.ca/pub/16-201-x/2009000/ct044-eng.htm [↑](#footnote-ref-2)