COMP 208

Group 4

Design Walkthrough

Contents

[Summary of Proposal 2](#_Toc34983375)

[Objectives 2](#_Toc34983376)

[Essential Features 2](#_Toc34983377)

[Desirable Features 2](#_Toc34983378)

[Specification Changes 2](#_Toc34983379)

[Interaction Chart 3](#_Toc34983380)

[UML Use Case Diagram (Main Features) 4](#_Toc34983381)

[Use-Case Descriptions 5](#_Toc34983382)

[Extended Use – Case Diagram 10](#_Toc34983383)

[App Layout Design 11](#_Toc34983384)

[App Navigation Chart 12](#_Toc34983385)

[Database Design 13](#_Toc34983386)

[Data Dictionary - Descriptions 13](#_Toc34983387)

[Global ER Diagram 13](#_Toc34983388)

[Data Dictionary - Relationships 14](#_Toc34983389)

[Data Dictionary - Attributes 14](#_Toc34983390)

[Global Logical Data Model 15](#_Toc34983391)

[Physical Table Structure 16](#_Toc34983392)

[Transaction Matrix (Data entry) 20](#_Toc34983393)

[Transaction Matrix (Deletion/Updates) 21](#_Toc34983394)

[Transaction Matrix (Data Queries) 22](#_Toc34983395)

[Pseudocode 23](#_Toc34983396)

[Evaluation Design: 27](#_Toc34983397)

[Review Against Plan: 29](#_Toc34983398)

[Design Report - Responsibilities 29](#_Toc34983399)

[Necessary changes 30](#_Toc34983400)

[Plans for implementation 30](#_Toc34983401)

# Summary of Proposal

## Objectives

Our main objectives for this project are to give people a social platform where they can share and like recipes, follow users who create recipes of interest and explore different cultures through cuisine, all while reducing food wastage through recipe suggestions based on ingredients people already have in their fridge.

## Essential Features

• Register as a base (standard) or pro user

• Browse recipes

• Add items to a shopping list

• Add and remove ingredients to virtual fridge

• Add and remove recipes if pro user

• Comment on recipes

• Follow users/add recipes to favorite

• Get suggestions /browse recipes based on items in the fridge

• Have a week’s top recipe/user section, in the home page

## Desirable Features

To expand the app beyond what we outlined in the specification we have a list of desirable features that we could add as a future update. These include things like:

• A smarter way to input ingredients (using your phones camera)

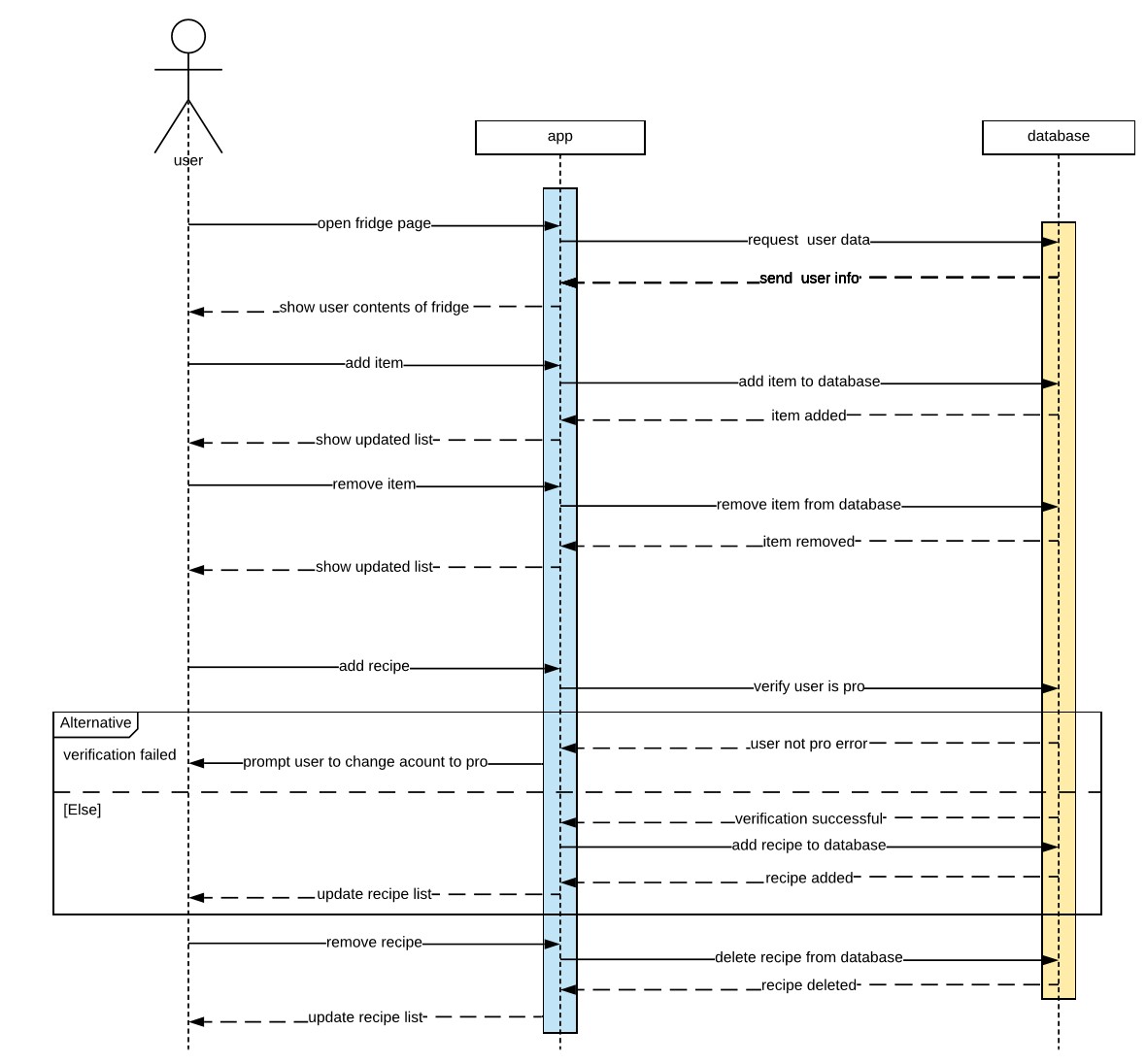
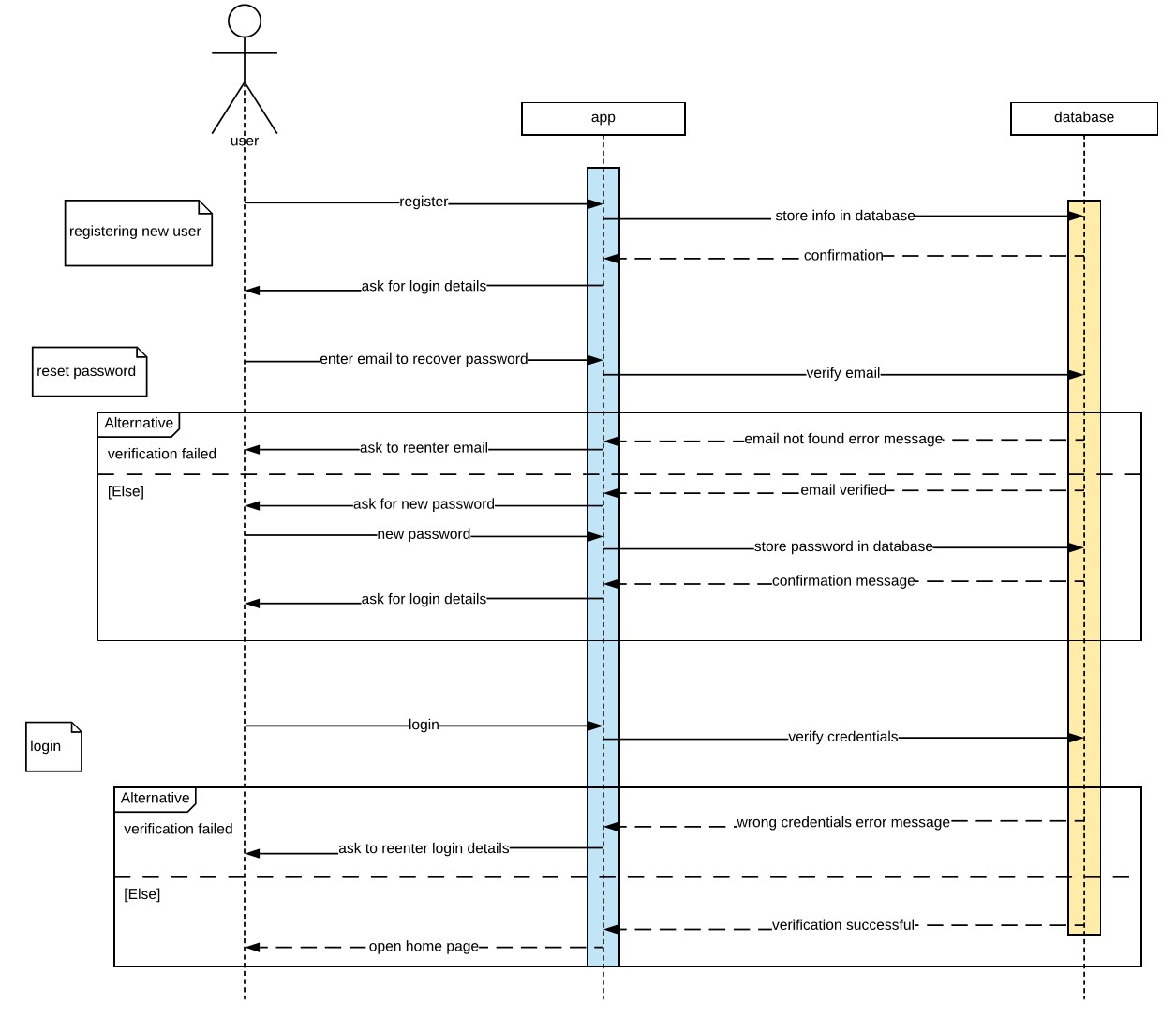
• A feature that allows users to get suggestions on how to swap certain foods in their diet for healthy alternatives.

## Specification Changes

As we were planning the development phase, we decided it would be a good idea to make the app start on the login page instead of the home page as we said in the specifications. The reason for this change is that there is not much a user can do on the app if they are not registered.

# Interaction Chart

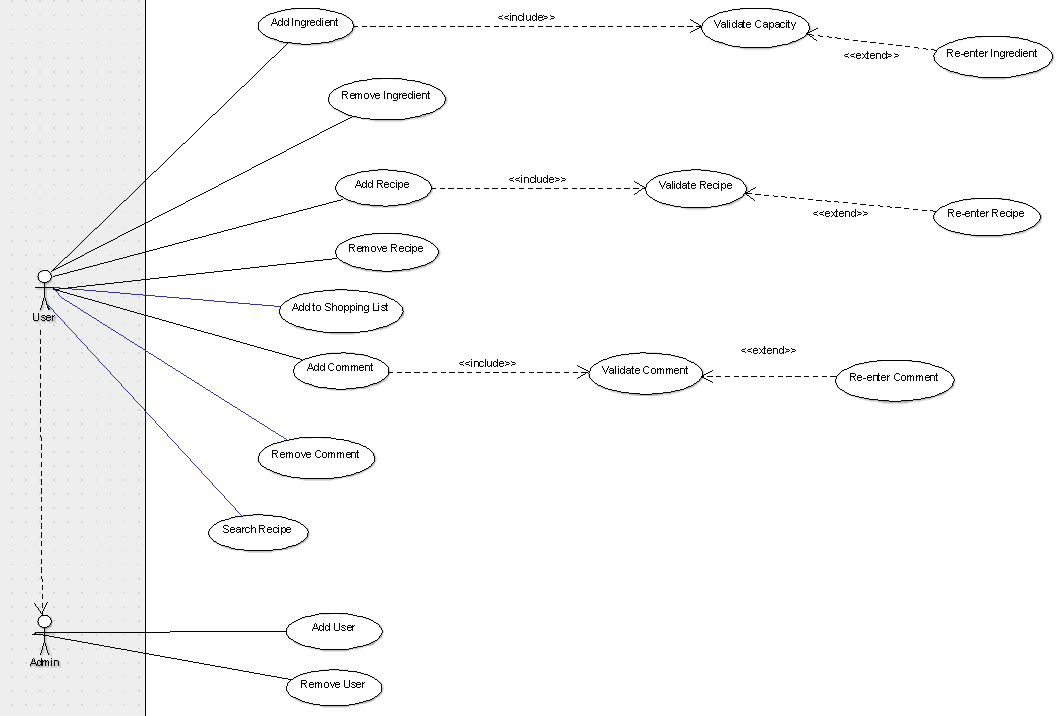
Registration, password recovery and login.



Add remove ingredients and recipes.

These diagrams show the interactions of the user with the app and how their actions would affect the database.

# UML Use Case Diagram (Main Features)



These use cases have been described on the next page

# Use-Case Descriptions

|  |  |
| --- | --- |
| **ID** | UC1 |
| **Name** | Add Ingredient |
| **Description** | User adds ingredient to Virtual Fridge |
| **Pre-Condition** | Account exists and initial fridge configuration done |
| **Event Flow** | 1. Enter Ingredient 2. Include UC2 ‘Validate Ingredient’ 3. If successful, add ingredient, else include UC3 ‘Re-enter Ingredient’ |
| **Extension Points** | Validate Ingredient  Re-enter Ingredient |
| **Triggers** | User selected the “Add Ingredient” option |
| **Post Condition** | Fridge is updated, notification is given to user, user is shown fridge |

|  |  |
| --- | --- |
| **ID** | UC2 |
| **Name** | Validate Ingredient |
| **Description** | User enters ingredient – validated to ensure it exists in DB |
| **Pre-Condition** | User enters ingredient |
| **Event Flow** | 1. Check Ingredient 2. If exists, add to Fridge and exit 3. Else, UC3, re-enter ingredient |
| **Extension Points** | Re-enter Ingredient |
| **Triggers** | User added a new ingredient |
| **Post Condition** | Fridge updated or user directed to re-enter UC3 |

|  |  |
| --- | --- |
| **ID** | UC3 |
| **Name** | Re-enter Ingredient |
| **Description** | User enters invalid ingredient and must re-enter a valid one |
| **Pre-Condition** | Ingredient entered is invalid |
| **Event Flow** | 1. Re-enter ingredient 2. UC2 – check ingredient again |
| **Extension Points** | Validate Ingredient |
| **Triggers** | User entered invalid ingredient |
| **Post Condition** | Balance updated, cash is given to customer, customer is logged out |

|  |  |
| --- | --- |
| **ID** | UC4 |
| **Name** | Remove Ingredient |
| **Description** | User wants to remove ingredient |
| **Pre-Condition** | User has ingredients in fridge |
| **Event Flow** | 1. Check which item user wants to remove 2. See if it exists in fridge and if it does remove it |
| **Extension Points** |  |
| **Triggers** | User selected the “Remove Ingredient” option |
| **Post Condition** | Fridge updated, and the fridge is shown to the user |

|  |  |
| --- | --- |
| **ID** | UC5 |
| **Name** | Add Recipe |
| **Description** | User wants to upload a recipe |
| **Pre-Condition** | Account exists |
| **Event Flow** | 1. Allow user to enter recipe 2. UC6 – check for validity 3. If fine, upload recipe and show user 4. Else, UC7 re-enter recipe |
| **Extension Points** | Validate Recipe  Re-enter recipe |
| **Triggers** | User selected the “Add Recipe” option |
| **Post Condition** | Recipe DB updated; recipe is shown to user |

|  |  |
| --- | --- |
| **ID** | UC6 |
| **Name** | Validate Recipe |
| **Description** | User enters recipe – validated to ensure it does not already exist in DB |
| **Pre-Condition** | User enters recipe |
| **Event Flow** | 1. Check recipe 2. If not exists, add to recipe DB and exit 3. Else, UC7, re-enter recipe |
| **Extension Points** | Re-enter recipe |
| **Triggers** | User added a new recipe |
| **Post Condition** | DB updated or user directed to re-enter UC7 |

|  |  |
| --- | --- |
| **ID** | UC7 |
| **Name** | Re-enter recipe |
| **Description** | User enters invalid recipe and must re-enter a valid one |
| **Pre-Condition** | recipe entered is invalid |
| **Event Flow** | 1. Re-enter recipe 2. UC6 – check recipe again |
| **Extension Points** | Validate recipe |
| **Triggers** | User entered invalid recipe |
| **Post Condition** | DB updated or user directed to re-enter UC7 |

|  |  |
| --- | --- |
| **ID** | UC8 |
| **Name** | Remove recipe |
| **Description** | User wants to remove recipe |
| **Pre-Condition** | User has recipe in fridge |
| **Event Flow** | 1. Check which recipe user wants to remove 2. See if it exists in DB and if it does remove it |
| **Extension Points** |  |
| **Triggers** | User selected the “Remove recipe” option |
| **Post Condition** | DB updated, and the confirmation is shown to the user |

|  |  |
| --- | --- |
| **ID** | UC9 |
| **Name** | Add to Shopping List |
| **Description** | User wants to add to shopping list |
| **Pre-Condition** | Account exists |
| **Event Flow** | 1. Allow user to enter item |
| **Extension Points** |  |
| **Triggers** | User selected the “Add to SL option” option |
| **Post Condition** | Shopping list updated; list is shown to user |

|  |  |
| --- | --- |
| **ID** | UC10 |
| **Name** | Add comment |
| **Description** | User wants to add a comment |
| **Pre-Condition** | Account exists |
| **Event Flow** | 1. Allow user to enter comment 2. UC11 – check for validity (profanity etc) 3. If fine, upload comment and show user 4. Else, UC12 re-enter comment |
| **Extension Points** | Validate comment  Re-enter comment |
| **Triggers** | User selected the “Add comment” option |
| **Post Condition** | Recipe DB updated, comment is shown to user |

|  |  |
| --- | --- |
| **ID** | UC11 |
| **Name** | Validate comment |
| **Description** | User enters comment – validated to ensure it does not contain profanity |
| **Pre-Condition** | User enters comment |
| **Event Flow** | 1. Check comment 2. If good, add to recipe DB and exit 3. Else, UC12, re-enter comment |
| **Extension Points** | Re-enter comment |
| **Triggers** | User added a new comment |
| **Post Condition** | DB updated or user directed to re-enter comment UC12 |

|  |  |
| --- | --- |
| **ID** | UC12 |
| **Name** | Remove Comment |
| **Description** | User/Admin wants to remove comment |
| **Pre-Condition** | User/Admin selects remove comment and admin rights if deleting another users comment |
| **Event Flow** |  |
| **Extension Points** |  |
| **Triggers** | User wants to remove comment |
| **Post Condition** | DB updated |

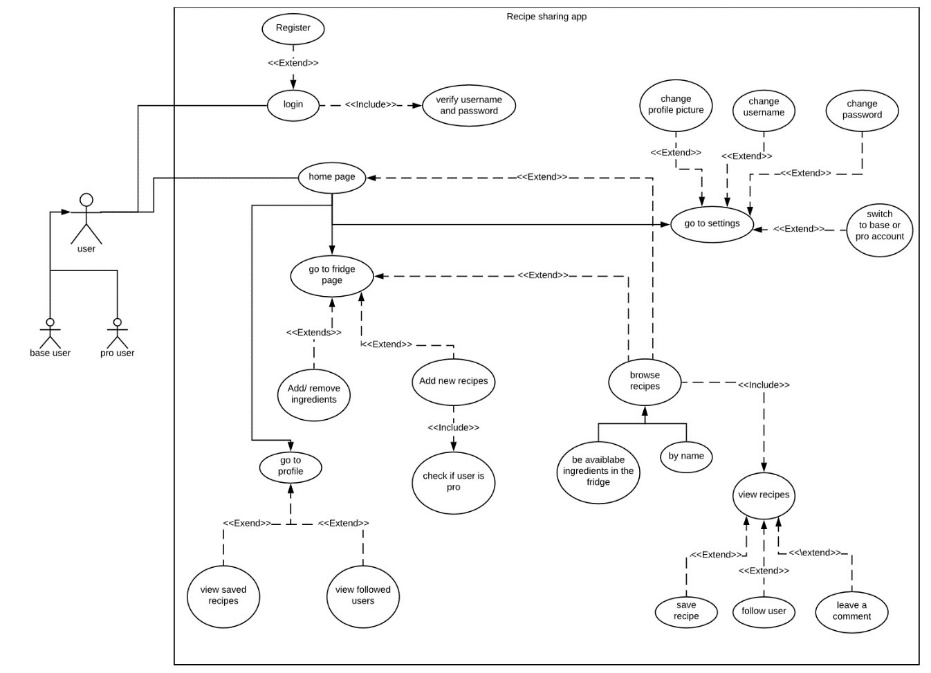
|  |  |
| --- | --- |
| **ID** | UC13 |
| **Name** | Search Recipe |
| **Description** | User wants to search for a recipe |
| **Pre-Condition** | User taps on search bar |
| **Event Flow** | 1. Search DB based on search term/items in virtual fridge 2. Show search results to user |
| **Extension Points** |  |
| **Triggers** | User wants to search recipes |
| **Post Condition** | Display search |

|  |  |
| --- | --- |
| **ID** | UC13 |
| **Name** | Search Recipe |
| **Description** | User wants to search for a recipe |
| **Pre-Condition** | User taps on search bar |
| **Event Flow** | 1. Search DB based on search term/items in virtual fridge 2. Show search results to user |
| **Extension Points** |  |
| **Triggers** | User wants to search recipes |
| **Post Condition** | Display search |

|  |  |
| --- | --- |
| **ID** | UC14 |
| **Name** | Add User |
| **Description** | Allows admin to add user |
| **Pre-Condition** | Admin taps on add user and they have admin rights |
| **Event Flow** | 1. Enter email 2. Search DB based on user to check they don’t already exist 3. Input user details 4. Display confirmation message |
| **Extension Points** |  |
| **Triggers** | Admin wants to add user |
| **Post Condition** | Display user account details and confirmation |

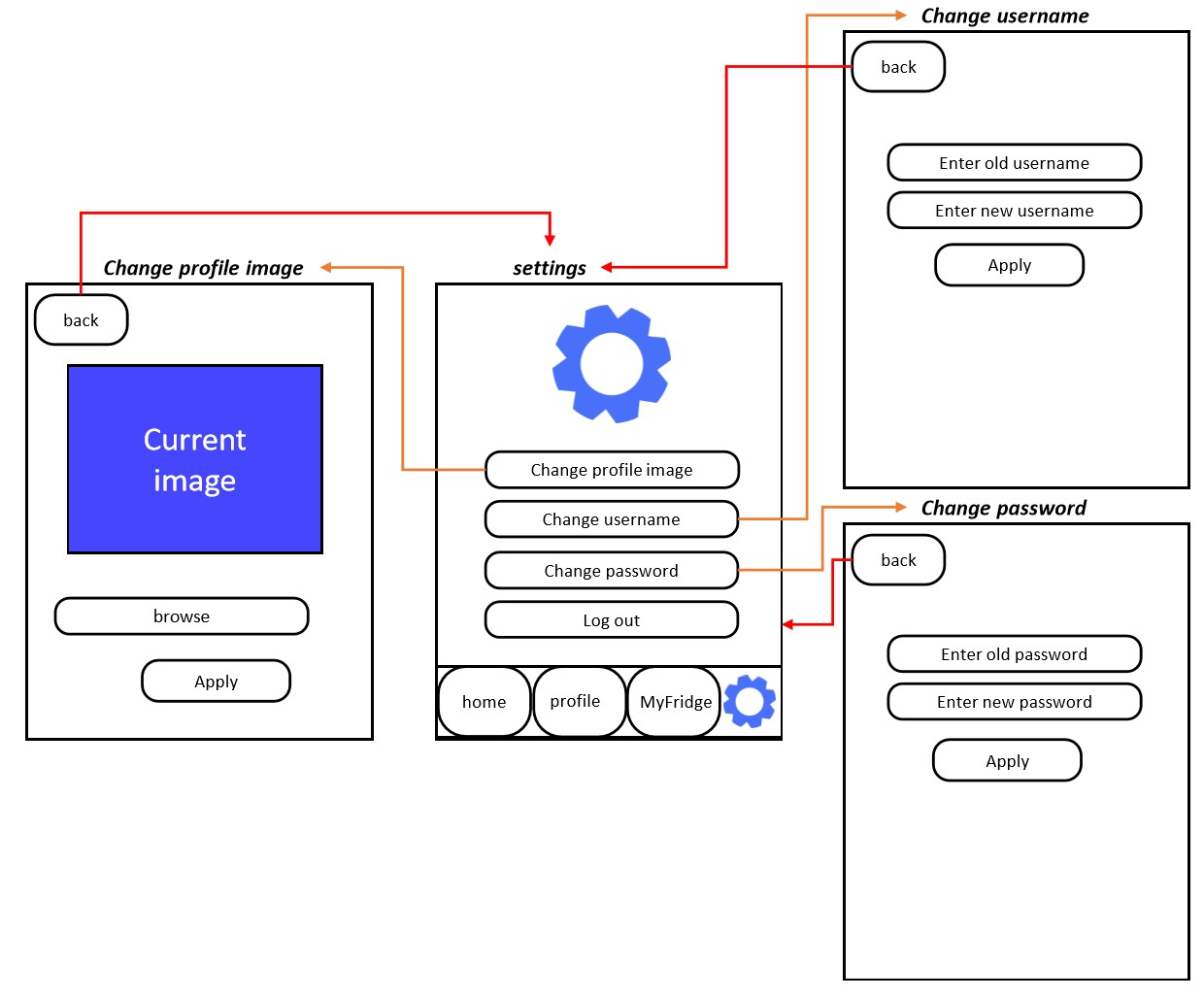
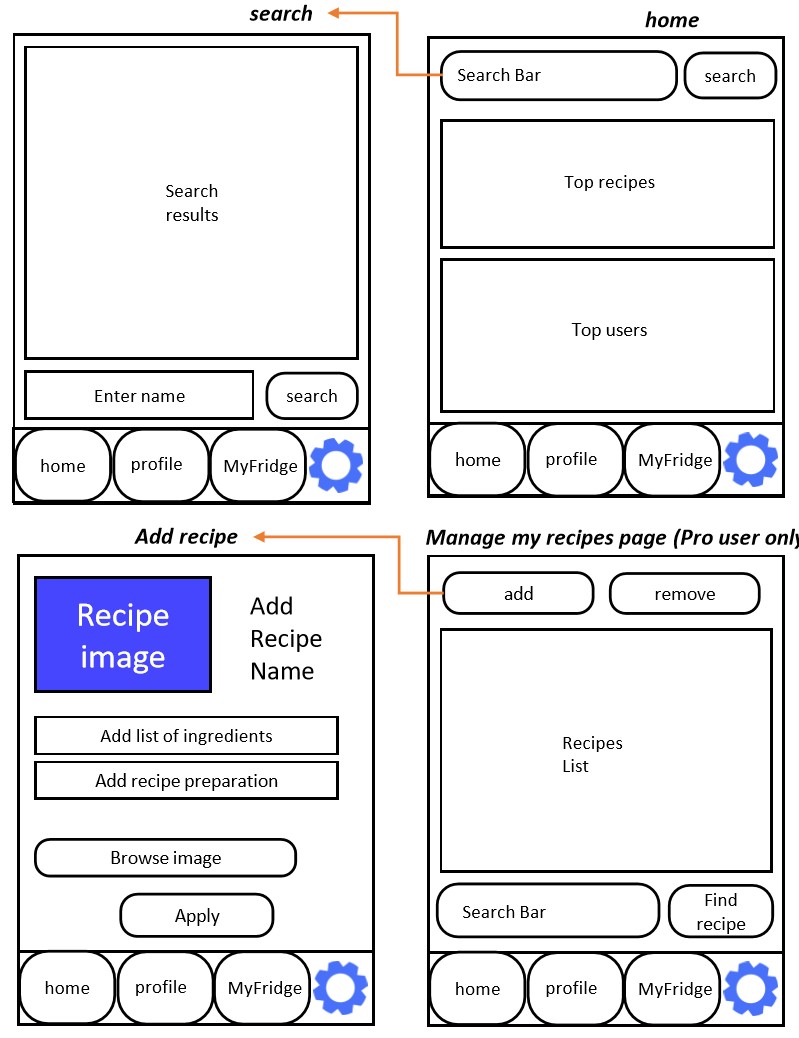
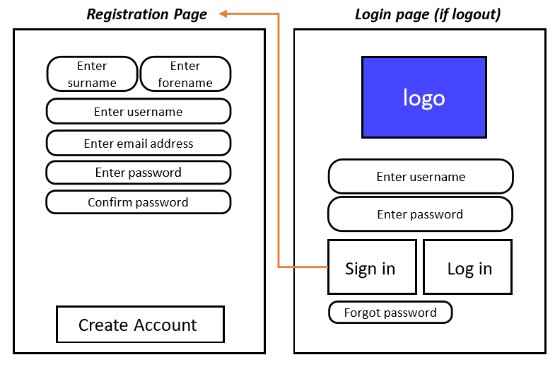
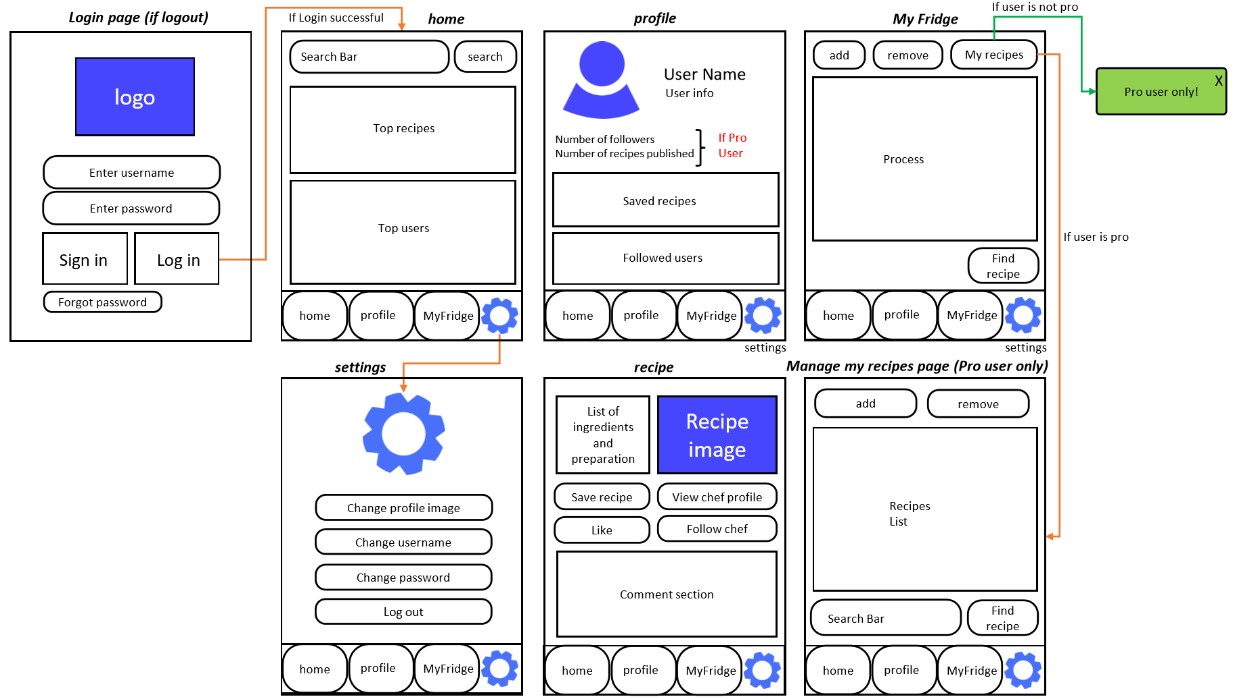
|  |  |
| --- | --- |
| **ID** | UC15 |
| **Name** | Remove recipe |
| **Description** | Allows admin to remove a user |
| **Pre-Condition** | Admin taps on remove user |
| **Event Flow** | 1. Enter email 2. Search DB based on user to check they already exist 3. Display confirmation message of deletion |
| **Extension Points** |  |
| **Triggers** | Admin wants to delete user |
| **Post Condition** | Display confirm message |

# Extended Use – Case Diagram



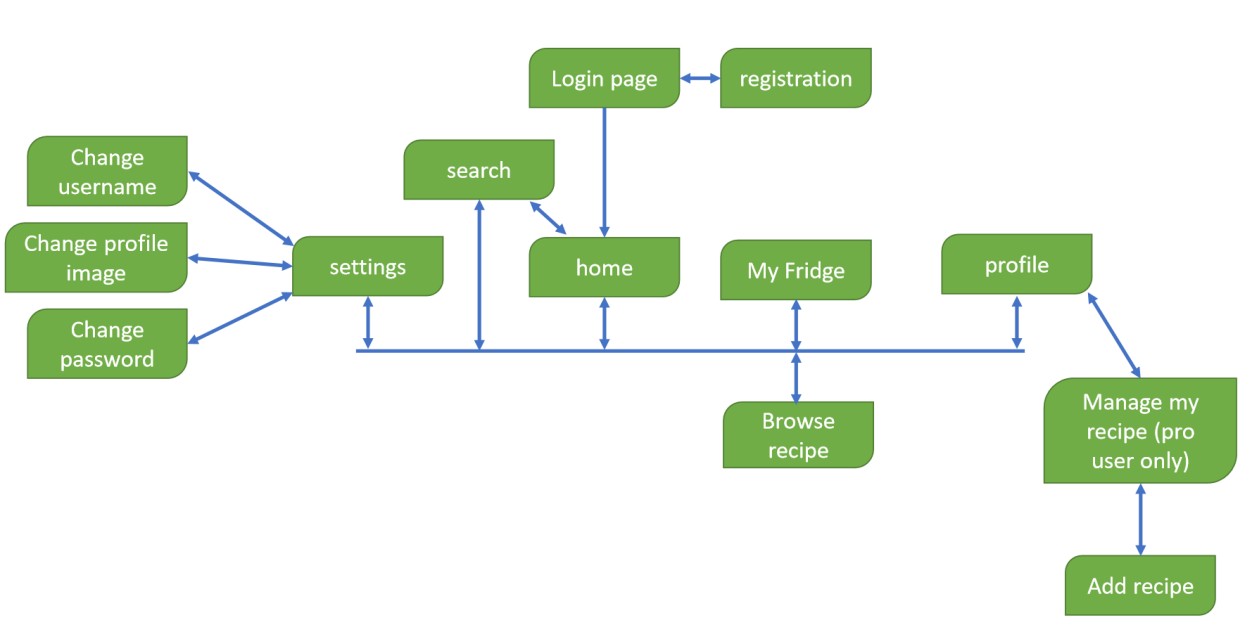
This use-case diagram differs from the one above as it shows more of an indication of the complete user experience and walkthrough of the different features, they might use throughout using the app.

# App Layout Design



# App Navigation Chart

Start

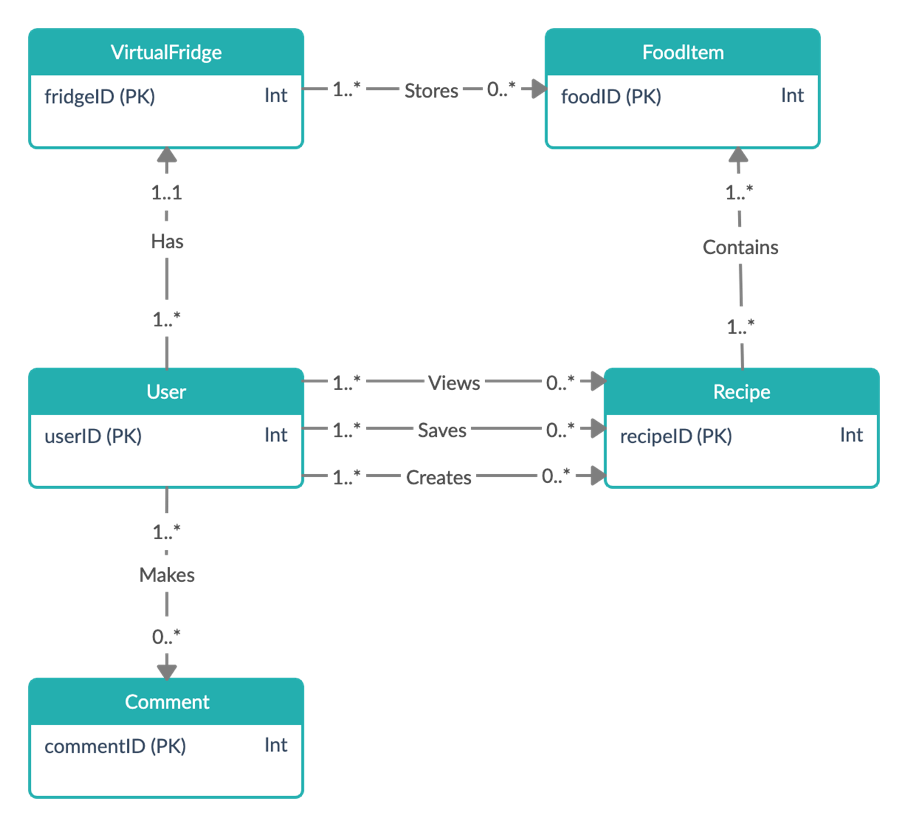


**Database Design**

Note: Originally for the design stage we structured our database following the SQL design methods. However, during implementation, we decided do use a NoSQL structure via Firebase to implement our database instead. This was because Firebase integrates seamlessly with Android Studio and the Firebase console provides great features to read, view, edit and maintain the database. Thus, the SQL ‘entities’ became NoSQL collections and records/rows became documents and the overall design is really similar to the original SQL based design (all relationships are still the same, just the technicalities in implementing them vary) so we have also included this design in the portfolio alongside the NoSQL design that was actually used in the implementation - which you will see later.

**Data Dictionary - Descriptions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Entity name** | **Description** | **Aliases** | **Occurrence** |
| VirtualFridge | Store the food items of the user | Fridge | Virtual fridge page – each user has a virtual fridge |
| FoodItem | Describe details regarding an item of food | Food, product, item | In the virtual fridge, in a recipe |
| User | Store details of the person using the application | Person, chef | Throughout the application - to use the application a ‘User’ must be defined |
| Recipe | Describe and store recipes | Meal | Recipes page, profile page, virtual fridge page |
| Comment | Store comments made by users regarding a recipe |  | Recipes page |

**Global ER Diagram**

**Data Dictionary – Relationships**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Multiplicity** | **Relationship** | **Multiplicity** | **Entity name** |
| VirtualFridge | 1..\* | Stores | 0..\* | FoodItem |
| User | 1..\* | Has | 1..1 | VirtualFridge |
| Recipe | 1..\* | Contains | 1..\* | FoodItem |
| User | 1..\* | Makes | 0..\* | Comment |
| User | 1..\* | Views | 0..\* | Recipe |
| User | 1..\* | Saves | 0..\* | Recipe |
| User | 1..\* | Creates | 0..\* | Recipe |

**Data Dictionary – Attributes**

\*No attributes are multi-valued

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entity name** | **Attributes** | **Data type and length** | **Description** | **Nulls** |
| VirtualFridge | **fridgeID (PK)**  *foodID (FK)*  quantity | int(6) fixed  int(7) fixed  int(3) variable | Uniquely identify a virtual fridge  Id of the food item in the fridge  Quantity of a food item in the fridge | No  No  No |
| FoodItem | **foodID (PK)**  name  calories  image  isVegan  isVegetarian  isPescatarian | int(7) fixed  varchar(50)  int(5) variable  varchar(50)  boolean  boolean  boolean | Uniquely identify a food item  Name of the food  Number of calories of the food  Directory of the image of the food  If the food is suitable for vegans  If the food is suitable for vegetarians  If the food is suitable for a pescatarian | No  No  Yes  Yes  No  No  No |
| User | **userID(PK)**  *fridgeID (FK)*  username  fName  lName  email  isPro  isVegan  isVegetarian  isPescatarian  followersCount  publishedRecipesCount | int(6) fixed  int(6) fixed  varchar(20)  varchar(30)  varchar(30) varchar(50)  boolean  boolean  boolean  boolean  int(6) fixed  int(6) fixed | Uniquely identify a user  ID of the user’s VirtualFridge  User’s username  User’s first name  User’s last name  User’s email  Indicate is the user is a pro user  Indicate if the user is a vegan  Indicate if the user is a vegetarian  Indicate if the user is a pescatarian  Number of followers the user has  Number of user’s published recipes | No  No  No  No  No  No  No  No  No  No  Yes  Yes |
| Recipe | **recipeID (PK)**  *userID(FK)*  name  image  ingredients  instructions  viewCount  likeCount | int(7) fixed  int(6) fixed  varchar(50)  varchar(50)  varchar(max)  varchar(max)  int(7) variable  int(7) variable | Uniquely identify a recipe  User who created the recipe  Name of the meal  Directory of the image of the food  Ingredients required to make the meal  Instructions on how to make the meal  Amount the recipe has been viewed  Number of likes the recipe has | No  No  No  Yes  No  No  No  No |
| Comment | **commentID (PK)**  *recipeID (FK)*  *userID(FK)*  commentContent | int(7) fixed  int(7) fixed  int(6) fixed  varchar(200) | Uniquely identify a comment  ID of the recipe the comment was made on  ID of user who made the comment  The actual comment that was made | No  No  No  No |

**Global logical data model**

|  |
| --- |
| **VirtualFridge** (fridgeID, foodID, quantity)  **Primary Key:** fridgeID  **Foreign Key:** foodID **references** FoodItem(foodID) |
| **FoodItem** (foodID,name,calories**,** image**,** isVegan**,** isVegetarian**,** isPescatarian)  **Primary Key:** foodID |
| **User** (userID, fridgeID, username,fName**,** lName**,** email**,** isPro,isVegan**,** isVegetarian**,** isPescatarian)  **Primary Key:** userID  **Foreign Key:** fridgeID **references** VirtualFridge(fridgeID) |
| **Recipe** (recipeID,userID**,** name**,** image**,** ingredients**,** instructions**,** viewCount**,** likeCount)  **Primary Key:** recipeID  **Foreign Key:** userID **references** User(userID) |
| **Comment** (commentID, recipeID, username, commentContent)  **Primary Key:** commentID  **Foreign Key:** recipeID **references** Recipe (recipeID) |

**Physical table structure**

**Virtual Fridge**

**VirtualFridge**( fridgeID Fridge\_Identification NOT NULL AUTO\_INCREMENT,

foodID Food\_Identification NOT NULL,

quantity Food\_Quantity NOT NULL)

**Primary key:** fridgeID

**Foreign key** foodID References **FoodItem**(foodID) ON DELETE CASCADE

|  |  |
| --- | --- |
| **Domain** | **Data type and length** |
| Fridge\_Identification | Fixed length integer, length 6 |
| Food\_Identification | Fixed length integer, length 7 |
| Quantity | Variable length integer, max length 3 |

**FoodItem**

**FoodItem** ( foodID Food\_Identification NOT NULL AUTO\_INCREMENT,

name Food\_Name NOT NULL,

calories Food\_Calories NULL,

image Food\_Image NULL

isVegan Food\_Vegan NOT NULL,

isVegetarian Food\_Vegetarian NOT NULL,

isPescatarian Food\_Pescatarian NOT NULL)

**Primary Key:** foodID

|  |  |
| --- | --- |
| **Domain** | **Data type and length** |
| Food\_Identification | Fixed length integer, length 7 |
| Food\_Name | Variable length character, max length 50 |
| Food\_Calories | Variable length integer, max length 5 |
| Recipe\_Image | Variable length BLOB, max length |
| Food\_Vegan | Boolean |
| Food\_Vegetarian | Boolean |
| Food\_Pescatarian | Boolean |

**User**

**User** ( userID User\_Identification NOT NULL AUTO\_INCREMENT,

fridgeID Fridge\_Identification NOT NULL,

username Username NOT NULL,

fName First\_Name NOT NULL,

lName Last\_Name NOT NULL,

email Email\_Address NOT NULL,

isPro User\_Pro\_User NOT NULL

isVegan User\_Vegan NOT NULL,

isVegetarian User\_Vegetarian NOT NULL,

isPescatarian User\_Pescatarian NOT NULL,

followersCount Followers\_Count NOT NULL,

publishedRecipesCount Recipes\_Count NULL)

**Primary Key:** userID

**Foreign Key:** fridgeID **references** VirtualFridge(fridgeID) ON DELETE CASCADE

|  |  |
| --- | --- |
| **Domain** | **Data type and length** |
| User\_Identification | Fixed length integer, length 6 |
| Fridge\_Identification | Fixed length integer, length 6 |
| Username | Variable length character, max length 20 |
| First\_Name | Variable length character, max length 30 |
| Last\_Name | Variable length character, max length 30 |
| Email\_Address | Variable length character, max length 50 |
| User\_Pro\_User | Boolean |
| User\_Vegan | Boolean |
| User\_Vegetarian | Boolean |
| User\_Pescatarian | Boolean |
| Followers\_Count | Fixed length integer, length 6 |
| Recipes\_Count | Fixed length integer, length 6 |

**Recipe**

**Recipe** (recipeID Recipe\_Identification NOT NULL AUTO\_INCREMENT,

userID User\_Identification NOT NULL,

nameRecipe\_Name NOT NULL,

image Recipe\_Image NULL,

ingredients Recipe\_Ingredients NOT NULL,

instructions Recipe\_Instructions NOT NULL,

viewCount Recipe\_Views NOT NULL,

likeCount Recipe\_Likes NOT NULL)

**Primary Key:** recipeID

**Foreign Key:** userID **references** User(userID) ON DELETE CASCADE

|  |  |
| --- | --- |
| **Domain** | **Data type and length** |
| Recipe\_Identification | Fixed length integer, length 6 |
| User\_Identification | Fixed length integer, length 6 |
| Recipe\_Name | Variable length character, max length 50 |
| Recipe\_Image | Variable length character, max length 50 |
| Recipe\_Ingredients | Variable length character, max length |
| Recipe\_Instructions | Variable length character, max length |
| Recipe\_Views | Variable length integer, max length 6 |
| Recipe\_Likes | Variable length integer, max length 6 |

**Comment**

**Comment** (commentID Comment\_Identification NOT NULL AUTO\_INCREMENT,

recipeID Recipe\_Identification NOT NULL,

userID User\_Identification NOT NULL,

commentContent Comment’s\_Content NOT NULL,)

**Primary Key:** commentID

**Foreign Key:** recipeID **references** Recipe (recipeID) ON DELETE CASCADE

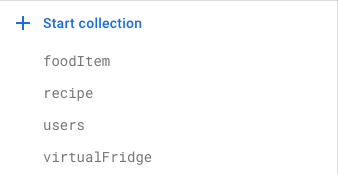
**Foreign Key:** userID **references** User (userID) ON DELETE CASCADE

|  |  |
| --- | --- |
| **Domain** | **Data type and length** |
| Comment\_Identification | Fixed length integer, length 7 |
| Recipe\_Identification | Fixed length integer, length 7 |
| User\_Identification | Fixed length integer, length 6 |
| Comment’s\_Content | Variable length character, max length 200 |

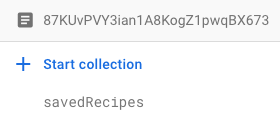
**Database Design – NoSQL**

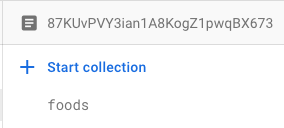
**NoSQL Collection Structure**

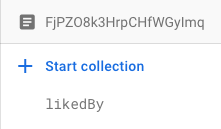
Instead of records/rows and primary keys our NoSQL collection structure has documents with unique ID’s.

**All Collections**

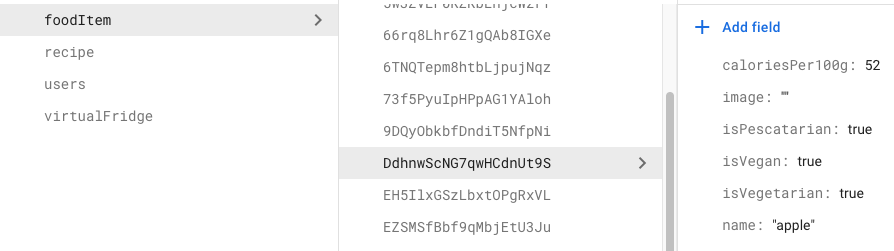
**All Sub-Collections**

****In users:

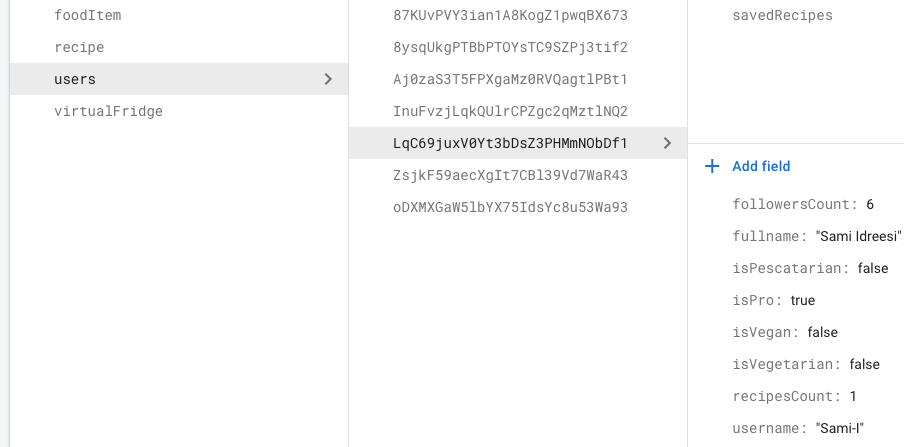
In virtualFridge:

In recipe:

**Individual Collections**

**FoodItem**

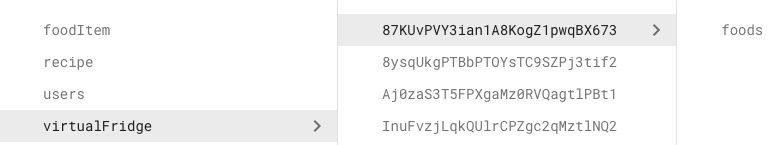
|  |  |
| --- | --- |
| **Field** | **Data type** |
| caloriesPer100g | Number |
| image | String |
| isPescatarian | Boolean |
| isVegan | Boolean |
| isVegetarian | Boolean |
| name | String |

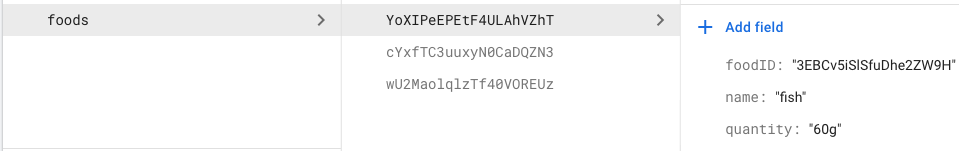
**Users**

|  |  |
| --- | --- |
| **Field** | **Data type** |
| followersCount | Number |
| fullName | String |
| isPro | Boolean |
| isPescatarian | Boolean |
| isVegan | Boolean |
| isVegetarian | Boolean |
| recipesCount | Number |
| username | String |

**Virtual Fridge**

Instead of storing attributes/fields of foodID and quantity like you would in an SQL table the virtualFridge collection contains a sub-collection which in turn stores the foods the user has added to their fridge. Also, every virtualFridge document has the same ID as the corresponding user to whom the virtualFridge is storing foods for as this is a one-to-one relationship.

****

**Foods**

|  |  |
| --- | --- |
| **Field** | **Data type** |
| foodID | String |
| name | String |
| quantity | String |

**Recipe**

|  |  |
| --- | --- |
| **Field** | **Data type** |
| ingredients | String |
| instructions | String |
| likeCount | Number |
| name | String |
| userID | String |
| viewCount | Number |

**Liked By**

|  |  |
| --- | --- |
| **Field** | **Data type** |
| likedByUser | String |

# 

# Transaction Matrix (Data entry)

1. Register new user

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge | X |  |  |  |
| users | X | X |  |  |
| recipe |  |  |  |  |
| foodItem |  |  |  |  |

Insertion into virtualFridge and users to create the corresponding document for the user and their virtualFridge and read for users to see if a user with the inputted username already exists

b) Adding new ingredient to Virtual Fridge

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge | X |  |  |  |
| users |  |  |  |  |
| recipe |  |  |  |  |
| foodItem | X (If ingredient doesn’t exist in foodItem collection) | X |  |  |

c) Uploading a new recipe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users |  |  | X |  |
| recipe | X |  |  |  |
| foodItem |  |  |  |  |

Insertion into recipe to add the new recipe and update for users to increment the published recipes count for the user

# Transaction Matrix (Deletion/Updates)

1. Update/Delete Recipe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users |  |  |  |  |
| recipe |  |  | X | X |
| foodItem |  |  |  |  |

1. Delete/Edit comment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users |  | X |  |  |
| recipe |  | X | X | X |
| foodItem |  |  |  |  |

1. Delete User

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  | X |
| users |  |  |  | X |
| recipe |  |  |  |  |
| foodItem |  |  |  |  |

1. Like a recipe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users |  |  |  |  |
| recipe |  |  | X |  |
| foodItem |  |  |  |  |

Update recipe to increment like count in the corresponding recipe document

1. Save a recipe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users | X |  |  |  |
| recipe |  | X |  |  |
| foodItem |  |  |  |  |

Read recipe to retrieve the recipe that needs saving and insert into users to add into the users savedRecipe sub-collection

1. Change username

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users |  | X | X (If username doesn’t already exist) |  |
| recipe |  |  |  |  |
| foodItem |  |  |  |  |

Read users to see if the username doesn’t already exist and update users to change the username

# Transaction Matrix (Data Queries)

1. List recipe details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users |  | X |  |  |
| recipe |  | X |  |  |
| foodItem |  |  |  |  |

Read recipe to retrieve the details of the recipe and read users to retrieve the username of the user who created the recipe from the userID field in recipe

1. List virtual fridge items/ingredients

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  | X |  |  |
| users |  |  |  |  |
| recipe |  |  |  |  |
| foodItem |  |  |  |  |

1. List users details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users |  | X |  |  |
| recipe |  |  |  |  |
| foodItem |  |  |  |  |

1. List saved recipes details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users |  | X |  |  |
| recipe |  | X |  |  |
| foodItem |  |  |  |  |

Read users to retrieve ID of saved recipe document in the users savedRecipe sub-collection and read recipe to retrieve the details of the saved recipe

1. List top users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | I | R | U | D |
| virtualFridge |  |  |  |  |
| users |  | X |  |  |
| recipe |  |  |  |  |
| foodItem |  |  |  |  |

# Pseudocode

compare method in settings (username/password)

pseudocode comments

k ← inputted string

n ← size of the array

A[n] ← elements taken from database for i in n

begin loop

e ← A[i]

if e is k then

report that the name is taken (1.)

break the loop

else if i is n and e is not k then

allow user to update his/her any information about them (2.)

end loop

1. describe that username or passowrd is already taken and go back to the change your password/username page

2. change the username or password of the user

Search bar

pseudocode comments

k ← string inputted

s ← size of string in characters C[s] ← characters in the string count ← 0

n ← 10 (1.)

S[n] ← strings that results from k that has been cut for i in s

begin loop

if C[i] is not space

word ← word + C[i]

else

S[count] ← word word ← empty string count ← count + 1

end loop

pass S[] to the search results

1. max size of passed array is 10

Search results

psuedocode comments

S[n] ← array of strings inputted in the searchbar s ← number of recipes in the database

R[s] ← array of names of recipes

UR[s] ← array which will store points with the search for recipes

SR[s][2] ← matrix which will be sorted for recipes c ← number of pro users in the database

P[c] ← array of pro users

UP[c] ← array which will store points with the search for pro users

SP[c][2] ← matrix which will be sorted for pro users for i in n (1.)

begin loop

for j in s begin loop

if S[i] is part of R[j] UR[j] ← UR[j] + 1

else if S[i] is ingredient of R[j] UR[j] ← UR[j] + 1

end loop end loop

for i in n (2.)

begin loop for j in c begin loop

if S[i] is part of P[j] UP[j] ← UP[j] + 1

else if S[i] is part of recipe made by P[j] UP[j] ← UP[j] + 1

end loop end loop

multiply each element of UR[] by the amount of views of R[]

max ← 0 (3.) ID ← 0

for i in s begin loop

for j in s begin loop

if max is smaller than UR[j] and is non-negative then

max ← UR[j] ID ← j

SR[i][0] ← max

SR[i][1] ← ID

end loop

1. loop which counts how much relevant is inserted string to the recipes’ names and ingridients inside of them.

2. loop which counts how relevant is inserted string to the pro users and their recipes.

3. the loop that sorts the recipes by how relevant their are to the inserted string (from the biggest to the smallest)

4. the loop that sorts the

pro users by how relevant they are to the inserted string (from the biggest to the smallest)

UR[SR[i][1]] ← -1 end loop

multiply each element of UP[] by the amount of followers of P[]

max ← 0 (4.) ID ← 0

for i in c begin loop

for j in c begin loop

if max is smaller than UP[j] and is non-negative then

max ← UP[j] ID ← j

SP[i][0] ← max

SP[i][1] ← ID end loop UP[SP[i][1]] ← -1

end loop

display the elements from SR[][] and SP[][] with the most likes that is not zero

Add ingredients to the fridge

psuedocode comments

s ← string inputted

a ← amount of the inputted item

l ← amount of food items in the database

FN[l] ← array that stores names of the food items c ← number of distinct products in the fridge

F[c] ← array of food items in the fridge (ID)

for i in l (1.)

begin loop

if s is FN[i]

s ← i end loop

for i in c begin loop

if s is c

increase the quantity of that item by a in the fridge else

add this item to the fridge with quantity of a end loop

1. loop to see if the items that person want to imput is in the database

Remove ingredients from the fridge

psuedocode comments s ← item user wants to remove (via event)

a ← amount of that item

c ← number of distinct products in the fridge F[c] ← array of food items in the fridge (ID) for i in c

begin loop

if ID of s is F[i]

remove the amount a of items F[i] from the fridge end loop

Find recipes based on the food items in the fridge

psuedocode comments s ← amount of all distinct items in the fridge

S[s] ← array of all of the items in the fridge (ID)

C[s] ← array of all of the items in the fridge (strings)

for i in s begn loop

C[i] ← name of the item S[i]

end loop

pass C[i] to the search result method

# Evaluation Design:

Below I have briefly described the main functionality that the system needs to be able to perform to be deemed a success. Even though there are many more functions we want our system to perform there are 5 main functionality criteria we need to test so that we can deem the system at least partly successful. As if these few functions can perform correctly the system will be functional for a main goal of giving people recipes based on what’s left in their fridge. Below I give a brief description about how each should perform and how we could test.

Each of tests briefly outlined below should be performed more than once with multiple different values so we can be sure that the system didn’t just fluke one of the tests and we can be sure it will actually work when it comes to our customers using it.

Can we add ingredients to our ‘virtual fridge’?

• This test will be a straight-forward feature to test. We will go onto the add ingredients section and try adding some ingredients of differing varieties. We will save the changes and go back to our current contents page and check that it has updated.

Can we add recipes to our recipe database?

• From the recipe page we should try to add a new recipe and save it, we should then return to the recipe search bar and see if it shows up when it is searched for.

Can we find those recipes on different devices?

• This a more important adaptation of the test above. Can we add recipes on one device and find them on another? This Requires us to create and save a new recipe and then search for it on a new device.

Can we get recipes shown to us by what ingredients we have?

• Now that we've tested that we can add ingredients and recipes (and search for them) we need to check that we can get recipes to show up based on what ingredients we have left.

When we cook a recipe do ingredients get deducted?

• A simple feature to test. When we've found a recipe, we are going to cook and pressed a button to confirm your cooking it your virtual fridge should have ingredients removed and be updated to represent what you have.

The testing will be done black box using members of our team that were not involved directly with the coding, or if that's not possible we will use coding members of the team but get them only test features they were not involved with as its likely our project is going to split into functions and these be completed by different sub-teams so we can use teams to test each other’s code.

We will be doing it black box as this means that we are purely testing its functionality and that it performs exactly how we wanted it to, giving us the correct outputs when used by

somebody who doesn’t know exactly how that function works.

Additionally, as our system is proposed to be used by the general public there are additional usability features, we need to test.

Is the system Intuitive?

• We want our system to be easy to use by any member of the general public so it’s important that the system is intuitive. Any should be able to pick up the app and within a few minutes understand exactly how it work and how to use it.

Are the buttons easy to press?

• We need to remember that the application is designed to be used on mobile devices with touchscreens. So, the buttons will need to be big so that they are easy to press.

Is the app easy to navigate?

• Our final app has multiple proposed features across different pages. It should be

easy for users to quickly navigate across these and gain access to each page, whether it’s through a menu or a navigation bar it needs to be easy to use and obvious which page you’re heading to.

Is the app nice to look at?

• Although un-important to how usable the app is it’s important that its visually appealing as this will mean more people are inclined to use it and therefore help it appeal to the masses, furthering our goal of reducing food wastage.

We will test all the above ideally using a focus group set up, we will get general members of the public ideally of varying age and different backgrounds so we can get an idea of the system performs for the entire public.

The setting of a focus group will be so that we can watch and see for ourselves exactly how the end users are interacting with the system as well as then receiving their feedback. The reason for this is being able to physically see where initial problems occur is much more effective than asking someone to describe it. Also, it ensures we get accurate feedback.

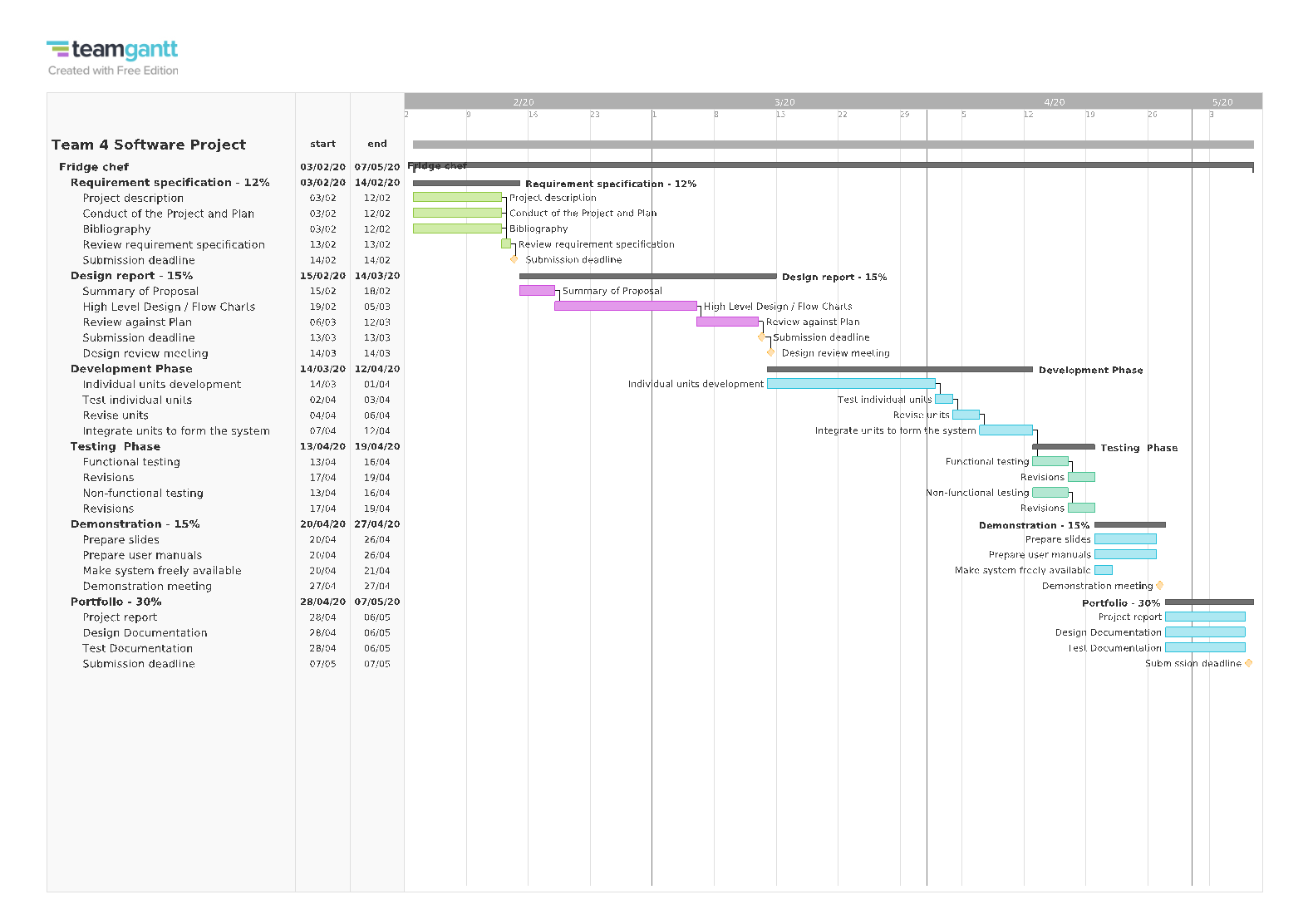
It’s important for these tests that we don’t use people on our course as their computer science background means they can’t be considered general public in this field. Additionally, we shouldn’t use anybody that we know as they may alter their feedback in order to be nice to us.

# Review Against Plan:

See below the Gantt Chart indicating the progress to date as of the design report and future objectives to complete.

Completed

Completed



# Design Report - Responsibilities

Aman – Summary of proposal, Transaction matrix, Use case descriptions and chart

Adam – Pseudocode

Sami – Data dictionaries, ER Diagrams, Logical table structures, Physical table structures

Saqab – Interface design, Use case diagram, Interaction chart, Tom – Evaluation design

Vlad – Interface design, navigation chart

# Necessary changes

All tasks set out in the plan and the Gantt chart have been completed so currently the project is on track for completion – provided that the next stages (development and testing) are also completed on time. So as of now there are no required changes.

# Plans for implementation

For the implementation of our system each group member will have a specific responsibility to aid in the development. This will be based on each member’s strengths or previous experiences.

To begin with, Vlad will work on designing the GUI layout of our app as he designed the user interface in the design section and has previous experience within this regard.

Sami will develop the login functionality to be used by Fridge Chef as he already made a start on this during the design and Tom will also aid in this by implementing the registration functionality.

Adam will implement the database based on the design and Sami will also aid in this as he was responsible for the database aspect in the design and can make any clarifications.

Aman and Saqab will work on implementing the various functions to be used by the application such as the process of suggesting meals based on the items in the fridge - because of their previous experience in iOS App Development. Further responsibilities regarding the implementation will be discussed during group meetings.

We also aim to maximize each member’s efficiency as when someone is finished with their current task they should move on to the next incomplete task or aid other members on the task they’re working on to speed the up the implementation phase.

We will use GitHub for version control to collaborate in the development. Members will work on the relevant branches based on the features or units of the application.