CS 411 Project Report - InvRec
Team 17 - PandaXpress

Team Members: Kanin Tangchartsiri (kanint2), Jingbin Cao (jingbin2), Gaozheng Liu (gl11)

Introduction into the Project:

Kitchen management and inventory management is an important part of keeping a clean and well maintained kitchen. This application is focused on allowing users to keep track of their ingredients in an organized manner. The users are able to see the ingredients which they have purchased at the store and are able to keep track of the expiry dates for all of them all in one centralized location. All the information regarding their current ingredients will be placed inside this centralized location, and in the process the users will also be able to look at recipes that they can build using the built in recipe finder that we have implemented.

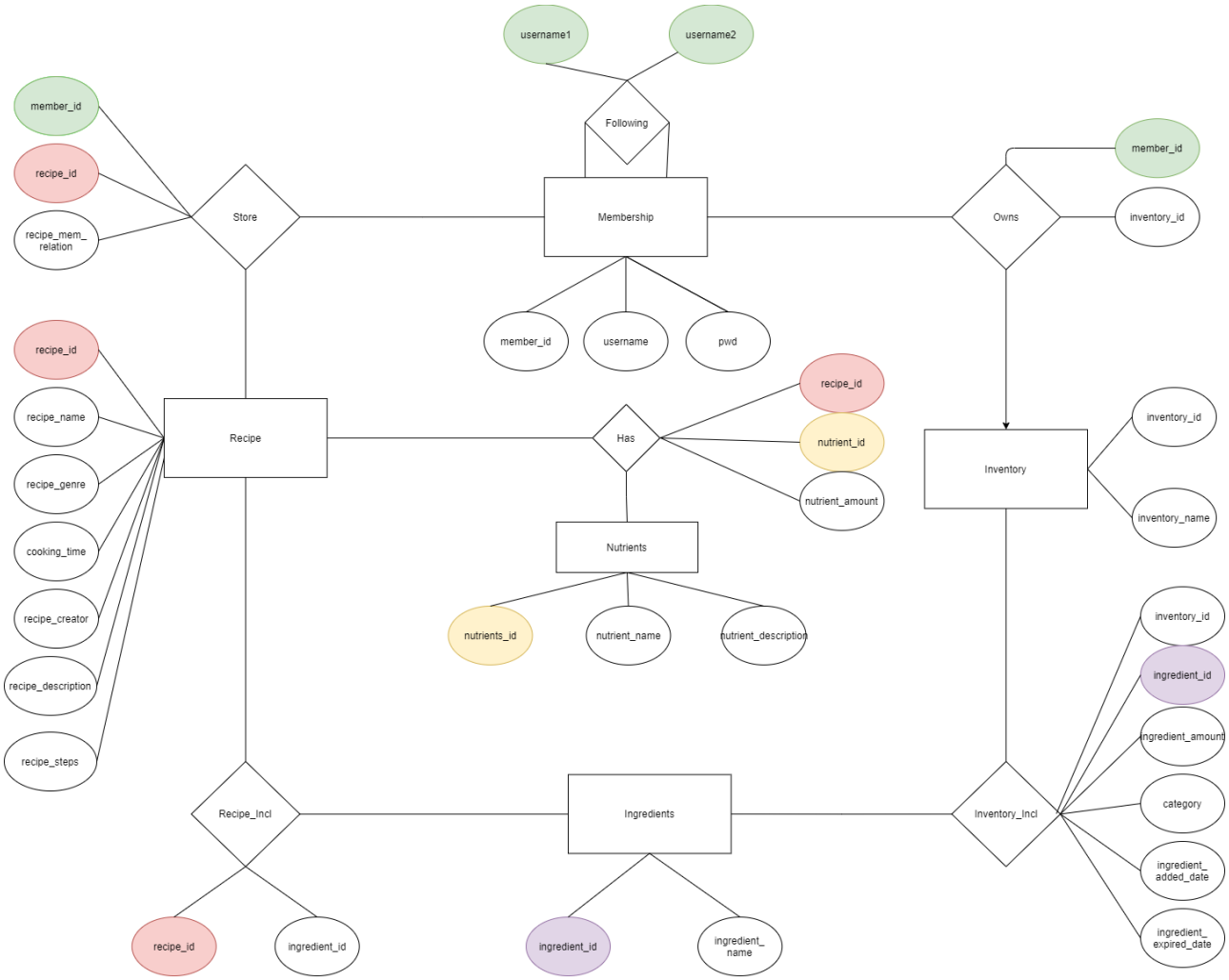
Usefulness of the Project:

As aforementioned, the primary usefulness of this project is that it allows for the users to centralize all their information about the ingredients they currently have. Typically users would have to search through their fridge in order to keep track of what ingredients they have. Additionally, checking for the expiration date on these ingredients requires the user to manually look at the labels every single time that they check. This application allows for all of this information to be ready at the fingertips of the user at one click of a button. The other problem that this application is able to solve is that users often struggle to find recipes that they can make with the current ingredients that they have. The recipe recommendation features that are integrated into this web application are able to automatically identify the recipes that match up the most with your current ingredients the most and provide you with the missing ingredients that you don't have in your pantry. As a result, this web application is able to serve as both an inventory management system and as an optimized cook book.

Introduction to the Databases:

The data that we are utilizing has been partitioned into different sections. We have recipe related data which contains the details including the ingredients, cooking time, nutritional value, steps, description etc. In particular, the nutrient value is directly linked to another database which stores the full nutritional value of each recipe which includes information regarding caloric intake and other necessary vitamins. From this we also have a database to store all of the ingredients. Mainly however, these ingredients are storing just the names and their id's. As for the user side of the application, we also have a database to store all the user's information. Namely this includes the username, password and the inventories that they own. This is then linked to the database of inventories that we have on hand. We are able to store the ingredients we have into the inventory include database which keeps track of all the ingredients that the users have placed into their respective inventories.

Updated ER:



Final Database DDLs and Index Design Analysis:

DDL:

CREATE TABLE Membership(
member_id INT PRIMARY KEY auto_increment,
username VARCHAR(50),
pwd VARCHAR(50)
);

CREATE TABLE Follow(
member_id INT(7),
following_id INT(7),
FOREIGN KEY(member_id) references Membership(member_id) ON DELETE CASCADE,
FOREIGN KEY(following_id) references Membership(member_id) ON DELETE CASCADE);
CREATE TABLE Inventory(

```
inventory_id INT auto_increment,
Inventory_name VARCHAR(50),
PRIMARY KEY(inventory_id)
);
create TABLE Recipes(
recipe_id int NOT NULL auto_increment,
recipe_creator int,
recipe_name varchar(1000),
recipe_genre varchar(1000),
recipe_description varchar(1000),
recipe_steps longtext,
cooking_time int DEFAULT NULL,
PRIMARY KEY (recipe_id),
FOREIGN KEY(recipe_creator) references Membership(member_id) ON DELETE SET NULL);

CREATE TABLE Owns(
member_id INT,
inventory_id INT,
FOREIGN KEY(member_id) references Membership(Member_id) ON DELETE CASCADE,
FOREIGN KEY(inventory_id) references Inventory(Inventory_id) ON DELETE CASCADE
);

CREATE TABLE Store(
recipe_id INT,
member_id INT,
recipe_member_rel VARCHAR(10),
FOREIGN KEY(member_id) references Membership(Member_id) ON DELETE CASCADE,
FOREIGN KEY(recipe_id) references Recipes(recipe_id) ON DELETE CASCADE
);

CREATE TABLE Nutrients(
nutrient_id INT1 PRIMARY KEY auto_increment,
nutrient_name VARCHAR(25),
nutrient_description longtext);
CREATE Table Has(
recipe_id INT,
nutrient_id INT1,
nutrient_amnt float4 default 0,
foreign key(recipe_id) references Recipes(recipe_id) ON DELETE CASCADE,
foreign key(nutrient_id) references Nutrients(nutrient_id) ON DELETE CASCADE
);
CREATE TABLE `Ingredients` (
`ingredient_id` int(11) NOT NULL auto_increment,
`ingredient_name` varchar(100) NOT NULL,
```

PRIMARY KEY (`ingredient_id`)
);
CREATE TABLE Recipe_Incl(
recipe_id INT,
ingredient_id INT,
FOREIGN KEY(ingredient_id) references Ingredients(ingredient_id) ON DELETE CASCADE,
FOREIGN KEY(recipe_id) references Recipes(recipe_id) ON DELETE CASCADE
);
CREATE TABLE `Inventory_Incl` (
`inventory_incl_id` int(11) NOT NULL auto_increment,
`inventory_id` int(11) NOT NULL,
`ingredient_id` int(11) NOT NULL,
`ingredient_amount` float NOT NULL,
`ingredient_unit` varchar(50) NOT NULL,
`ingredient_added_date` date NOT NULL,
`ingredient_expiry_date` date NOT NULL,
PRIMARY KEY (`inventory_incl_id`),
foreign key(inventory_id) references Inventory(inventory_id) ON DELETE CASCADE,
foreign key(ingredient_id) references Ingredients(ingredient_id) ON DELETE CASCADE
);

INDEX:

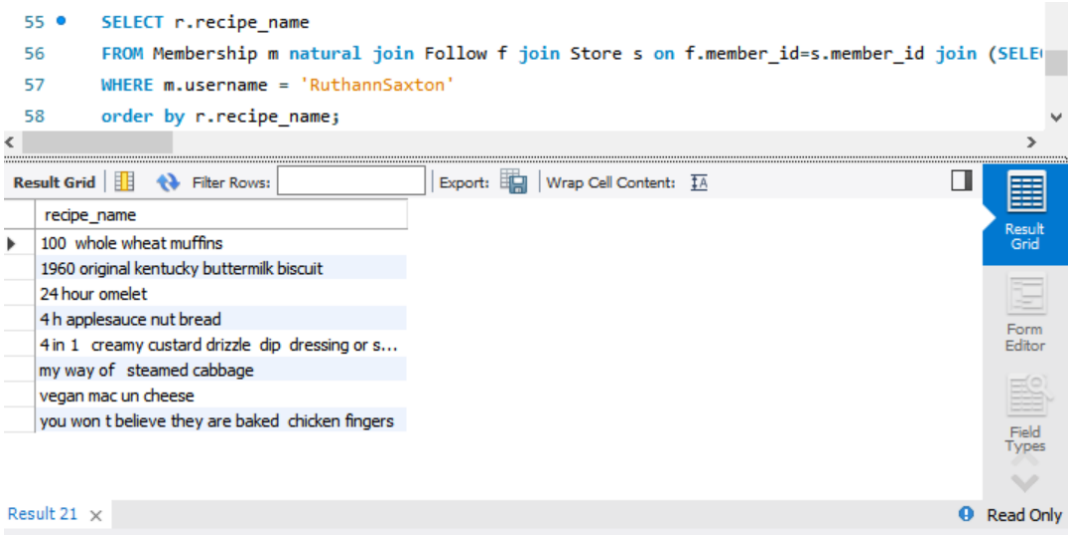
So the query that we've implemented an index on is the following:

SELECT r.recipe_name

FROM Membership m natural join Follow f join Store s on f.member_id=s.member_id join (SELECT FROM Recipes WHERE cooking_time < 100) AS r on s.recipe_id = r.recipe_id*

WHERE m.username = 'RuthannSaxton'

order by r.recipe_name;



INDEX Design: Recipes(cooking_time), Membership(username), and use both.

Due to the fact that we used foreign keys as much as possible, for join operations, we weren't able to find the useful indexes.

Original explain analysis result:

| -> Sort: <temporary>.recipe_name (actual time=0.108..0.108 rows=8 loops=1)

-> Stream results (actual time=0.046..0.087 rows=8 loops=1)

-> Nested loop inner join (cost=4.73 rows=0) (actual time=0.045..0.085 rows=8 loops=1)

-> Nested loop inner join (cost=4.62 rows=0) (actual time=0.038..0.068 rows=8 loops=1)

-> Nested loop inner join (cost=4.30 rows=1) (actual time=0.014..0.026 rows=10 loops=1)

-> Filter: ((s.member_id is not null) and (s.recipe_id is not null)) (cost=1.15 rows=9) (actual time=0.007..0.014 rows=10 loops=1)

-> Table scan on s (cost=1.15 rows=9) (actual time=0.006..0.011 rows=10 loops=1)

-> Filter: (m.username = 'RuthannSaxton') (cost=0.25 rows=0) (actual time=0.001..0.001 rows=1 loops=10)

-> Single-row index lookup on m using PRIMARY (member_id=s.member_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=10)

-> Filter: (Recipes.cooking_time < 100) (cost=0.29 rows=0) (actual time=0.004..0.004 rows=1 loops=10)

-> Single-row index lookup on Recipes using PRIMARY (recipe_id=s.recipe_id) (cost=0.29 rows=1) (actual time=0.004..0.004 rows=1 loops=10)

-> Index lookup on f using member_id (member_id=s.member_id) (cost=0.71 rows=1) (actual time=0.001..0.002 rows=1 loops=8)

1 row in set (0.16 sec)

1. Recipes(cooking_time)

```
mysql> CREATE INDEX idx_cookingtime ON Recipes(cooking_time);
Query OK, 0 rows affected (0.25 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

| -> Sort: <temporary>.recipe_name (actual time=0.099..0.100 rows=8 loops=1)
-> Stream results (actual time=0.047..0.086 rows=8 loops=1)
-> Nested loop inner join (cost=4.96 rows=1) (actual time=0.046..0.084 rows=8 loops=1)
-> Nested loop inner join (cost=4.62 rows=1) (actual time=0.039..0.067 rows=8 loops=1)
-> Nested loop inner join (cost=4.30 rows=1) (actual time=0.025..0.039 rows=10 loops=1)
-> Filter: ((s.member_id is not null) and (s.recipe_id is not null)) (cost=1.15 rows=9) (actual

time=0.016..0.023 rows=10 loops=1)
-> Table scan on s (cost=1.15 rows=9) (actual time=0.014..0.020 rows=10 loops=1)
-> Filter: (m.username = 'RuthannSaxton') (cost=0.25 rows=0) (actual time=0.001..0.001 rows=1 loops=10)
-> Single-row index lookup on m using PRIMARY (member_id=s.member_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=10)
-> Filter: (Recipes.cooking_time < 100) (cost=0.36 rows=1) (actual time=0.002..0.003 rows=1 loops=10)
-> Single-row index lookup on Recipes using PRIMARY (recipe_id=s.recipe_id) (cost=0.36 rows=1) (actual time=0.002..0.002 rows=1 loops=10)
-> Index lookup on f using member_id (member_id=s.member_id) (cost=0.40 rows=1) (actual time=0.001..0.002 rows=1 loops=8)
|
1 row in set (0.15 sec)

2. Membership(username)

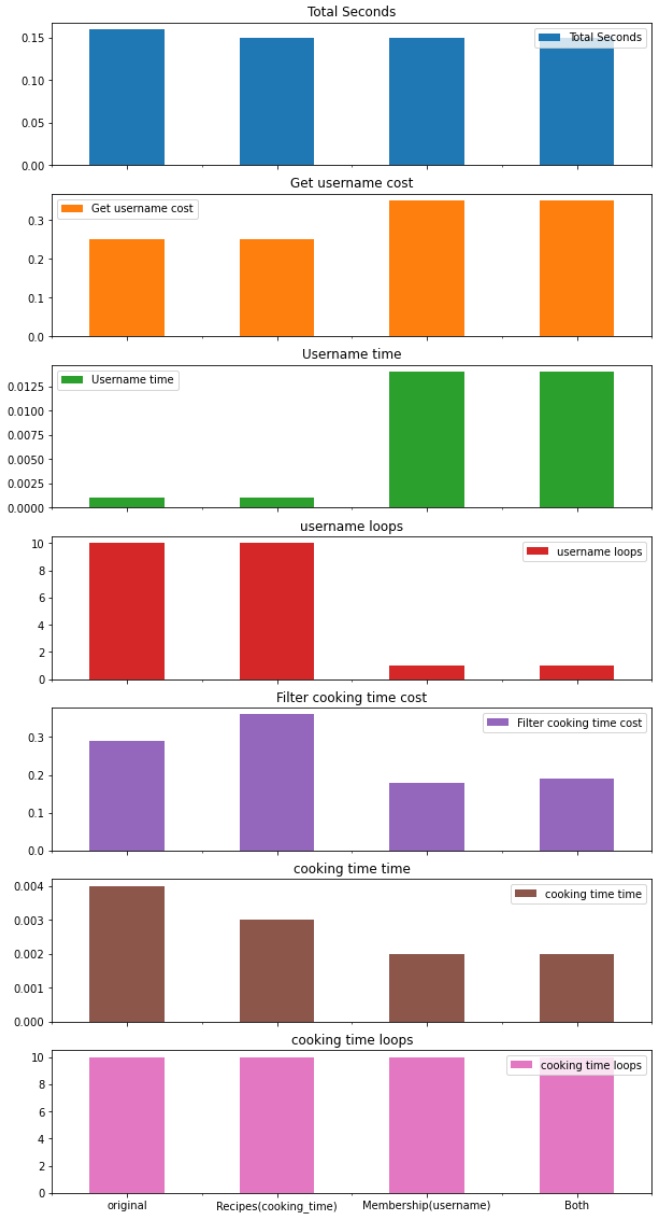
```
mysql> CREATE INDEX idx_username ON Membership(username);
Query OK, 0 rows affected (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

| -> Sort: <temporary>.recipe_name (actual time=0.092..0.093 rows=8 loops=1)
-> Stream results (actual time=0.053..0.078 rows=8 loops=1)
-> Nested loop inner join (cost=5.72 rows=4) (actual time=0.052..0.076 rows=8 loops=1)
-> Inner hash join (s.member_id = m.member_id) (cost=2.23 rows=12) (actual time=0.041..0.049 rows=10 loops=1)
-> Filter: (s.recipe_id is not null) (cost=0.83 rows=9) (actual time=0.006..0.012 rows=10 loops=1)
-> Table scan on s (cost=0.83 rows=9) (actual time=0.005..0.010 rows=10 loops=1)
-> Hash
-> Nested loop inner join (cost=0.74 rows=1) (actual time=0.024..0.027 rows=1 loops=1)
-> Index lookup on m using idx_username (username='RuthannSaxton') (cost=0.35 rows=1) (actual time=0.014..0.014 rows=1 loops=1)
-> Index lookup on f using member_id (member_id=m.member_id) (cost=0.39 rows=1) (actual time=0.009..0.012 rows=1 loops=1)
-> Filter: (Recipes.cooking_time < 100) (cost=0.18 rows=0) (actual time=0.002..0.002 rows=1 loops=10)
-> Single-row index lookup on Recipes using PRIMARY (recipe_id=s.recipe_id) (cost=0.18 rows=1) (actual time=0.002..0.002 rows=1 loops=10)
|
1 row in set (0.15 sec)

3. Both

| -> Sort: <temporary>.recipe_name (actual time=0.089..0.089 rows=8 loops=1)
-> Stream results (actual time=0.050..0.075 rows=8 loops=1)

-> Nested loop inner join (cost=5.72 rows=12) (actual time=0.049..0.073 rows=8 loops=1)
-> Inner hash join (s.member_id = m.member_id) (cost=2.23 rows=12) (actual time=0.038..0.046 rows=10 loops=1)
-> Filter: (s.recipe_id is not null) (cost=0.83 rows=9) (actual time=0.005..0.011 rows=10 loops=1)
-> Table scan on s (cost=0.83 rows=9) (actual time=0.004..0.009 rows=10 loops=1)
-> Hash
-> Nested loop inner join (cost=0.74 rows=1) (actual time=0.023..0.025 rows=1 loops=1)
-> Index lookup on m using idx_username (username='RuthannSaxton') (cost=0.35 rows=1) (actual time=0.014..0.014 rows=1 loops=1)
-> Index lookup on f using member_id (member_id=m.member_id) (cost=0.39 rows=1) (actual time=0.008..0.010 rows=1 loops=1)
-> Filter: (Recipes.cooking_time < 100) (cost=0.19 rows=1) (actual time=0.002..0.002 rows=1 loops=10)
-> Single-row index lookup on Recipes using PRIMARY (recipe_id=s.recipe_id) (cost=0.19 rows=1) (actual time=0.002..0.002 rows=1 loops=10)
|
1 row in set (0.15 sec)



	Total Seconds	Get username cost	User name time	username loops	Filter cooking time cost	cooking time time	cooking time loops
original	0.16	0.25	0.001	10	0.29	0.004	10
Recipes(cooking_time)	0.15	0.25	0.001	10	0.36	0.003	10

Membership(username)	0.15	0.35	0.014	1	0.18	0.002	10
Both	0.15	0.35	0.014	1	0.19	0.002	10

From the table and charts, we find that:

- Creating an index for username is quicker, and it also reduces the number of loops when searching the exact username. In this query it helps to reduce the time when filtering cooking time, but in other examples, they may not be related to each other.
- Creating an index for cooking time will reduce the time to filter recipes that meet the cooking time requirement.
- We decided to use BOTH based on the advantages of each index, even if the total time of using both indexes has not improved a lot from this example.

Source of Data:

https://www.kaggle.com/shuyangli94/food-com-recipes-and-user-interactions So this was the data source that we used to gather our recipe data. The data source itself has a lot of superfluous information that is not needed for our general purposes and so once we had downloaded the CSV files from Kaggle we utilized google collab to process the data further. We first selected recipes with id's starting from 200,000 - 210,000 from PP_recipes.csv, using pandas. This allowed us to streamline our dataset to optimize our run time as we upload the dataset to mysql workbench. With the extracted id's, we then extract the ingredients from those recipes. Within the PP_recipes.csv file there are ingredient id's for all of those involved. These were cross referenced to get the ingredient names.

The next piece of information that was important was the creation of temporary user information. For the inventories that we used for demo purposes, we manually created ten inventories and randomly assigned them id's. From this, we also manually assigned a list of ingredients into our inventory_incl table. This was done both through both the pandas library as well as manual manipulation using Microsoft Excel. The usernames and passwords were also randomly generated and assigned.

Application Design and Features:

- Inventory management, caloric intake and nutritional benefit of each ingredient.
- Recommend recipes based on the ingredients that you currently have as well as recommend recipes that can be made with 1-2 additional ingredients that you can go out and buy at the supermarket.
- Users can create their own recipes and update this information into their online recipe book.
- Users can like other recipes.
- Users can follow other users and subscribe to others recipes.

Advanced Features of the Database:

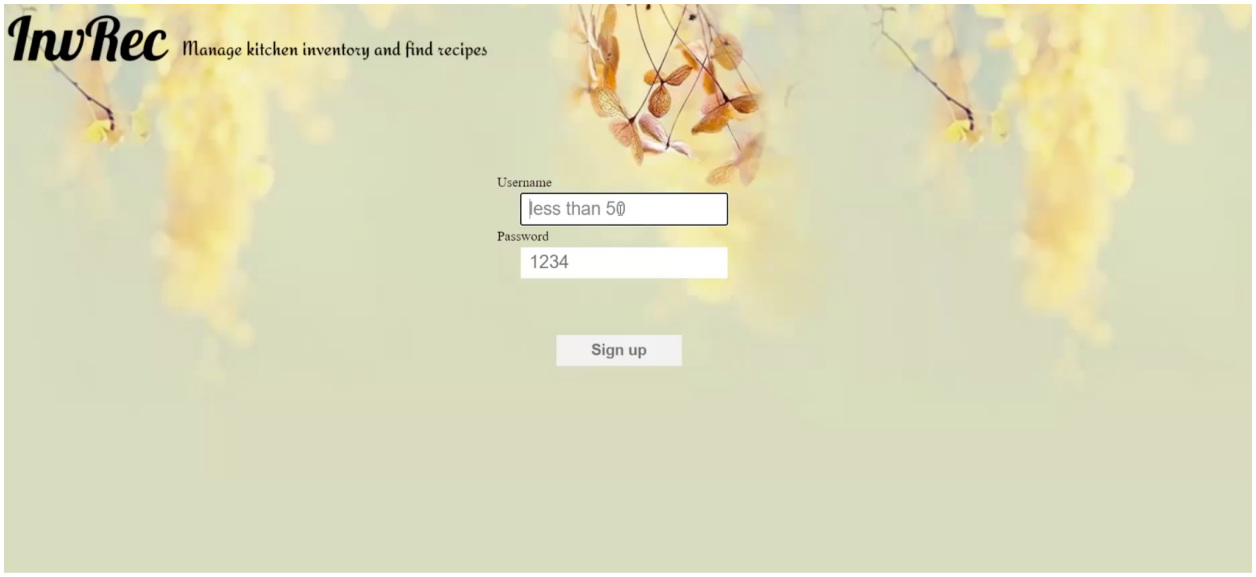
_____For the advanced database we chose a stored procedure + trigger. The triggers are generally very straight forward. We used it to implement a checker for when the expiry dates are not added properly. We also have a trigger to check for when an inventory is deleted. This makes sure that we actually delete the rest of the entries in the inventory incl as well as our own table. We chose a stored procedure as we wanted to output popular recipes. It was decided that a stored procedure would be best as we weren't necessarily updating the database but rather pulling information from different parts of the database to output onto one single page. This is why the stored procedure would help simplify the call back.

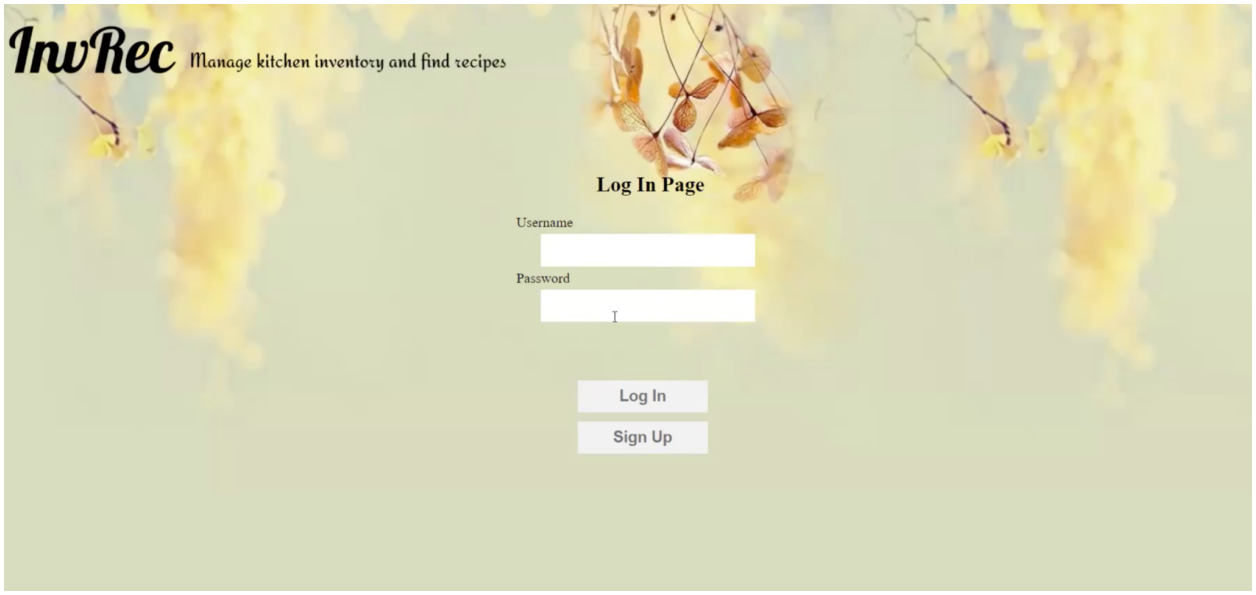
Our actual stored procedure involves grabbing popular recipes that are liked by multiple people. The procedure then filters out the popular recipes which don't cover the full nutrient requirements as well as filtering out recipes with long cooking time to get the best recipe that's both nutritious and convenient. Then we match the ingredients in the users inventory ingredients with the recipe ingredients. We then return the ingredients that are missing from those particular recipes so that the user can go out and buy them. This procedure allows us to provide users with the most popular and nutritious recipes that match up the best with the user's current state of their inventory.

We have multiple processes that we want to perform and so using a stored procedure is the best method for this particular functionality of our project.

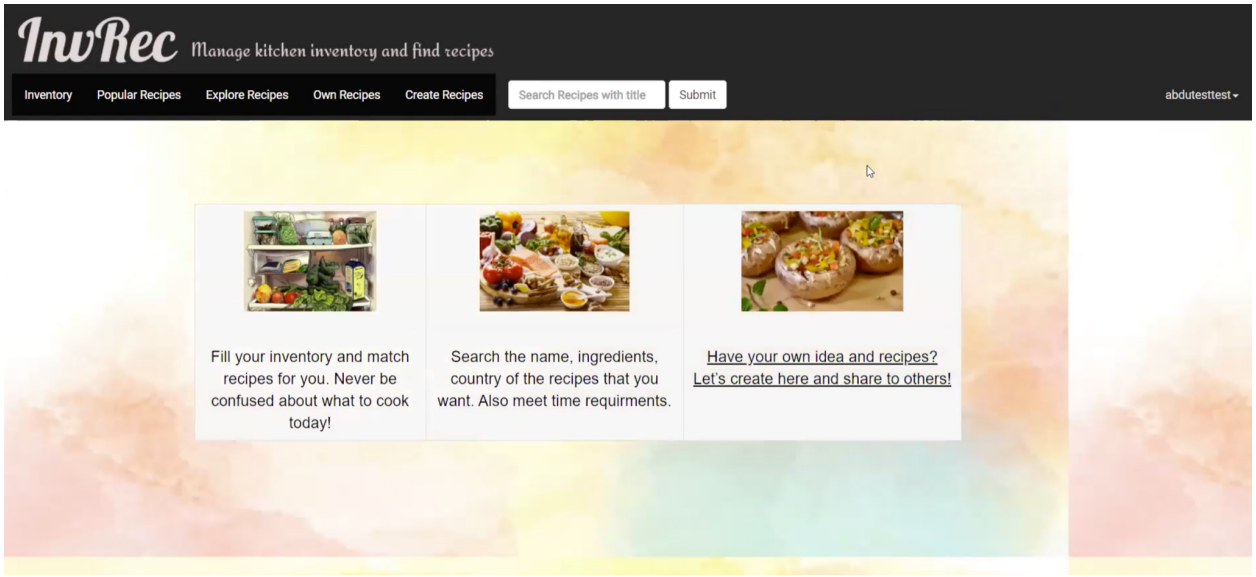
Dataflow:

In the beginning, user can sign up and then sign in into the system.

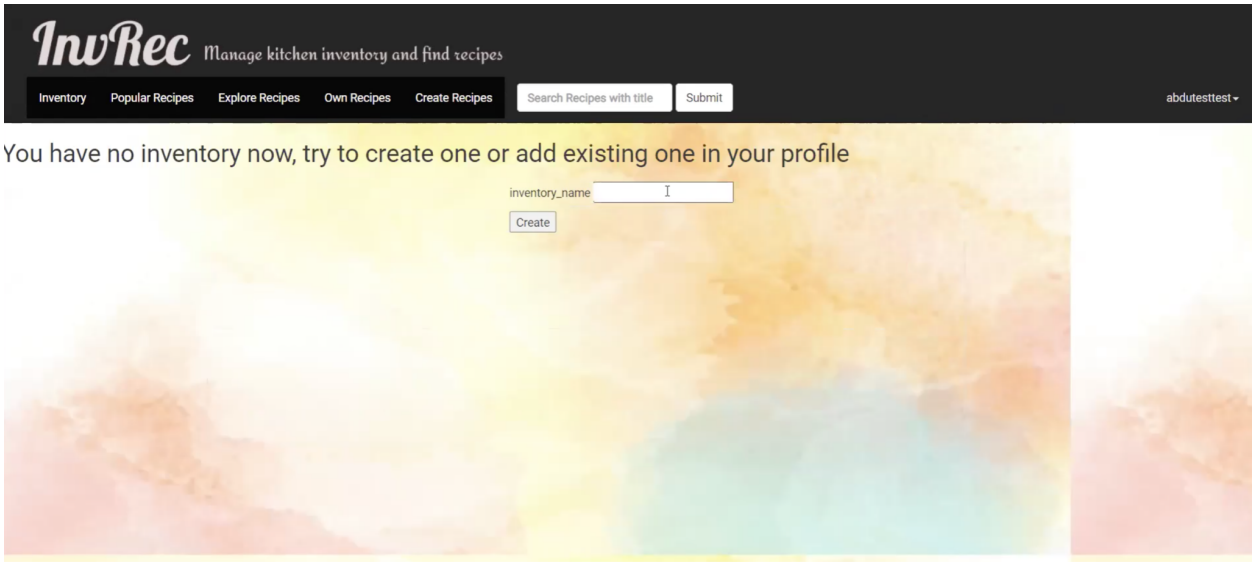




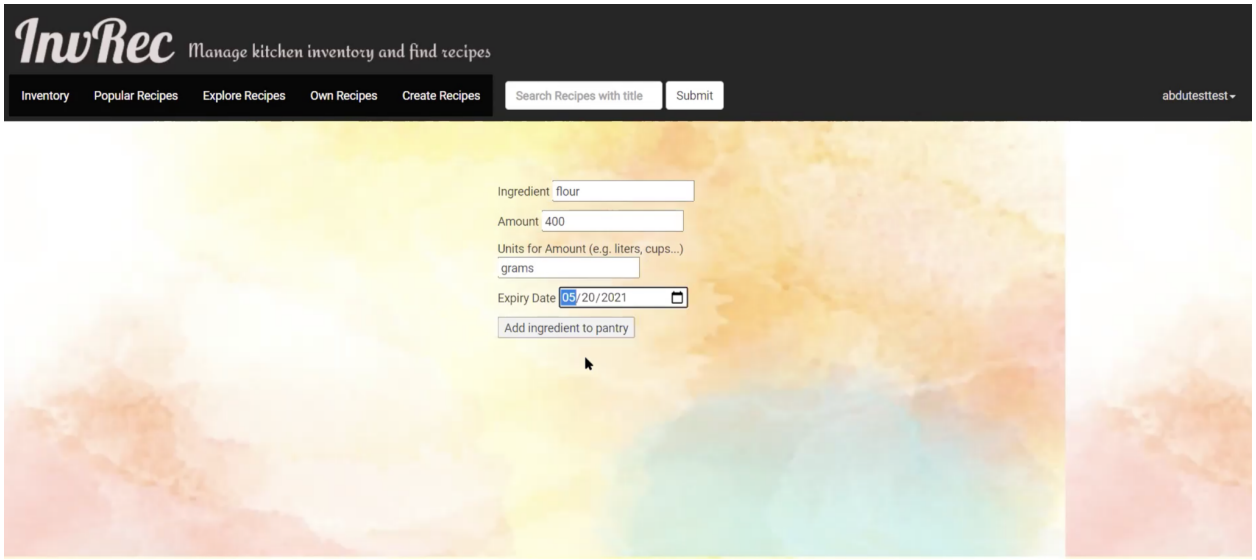
After sign in, the user abudutesttest will be directed to the home page.

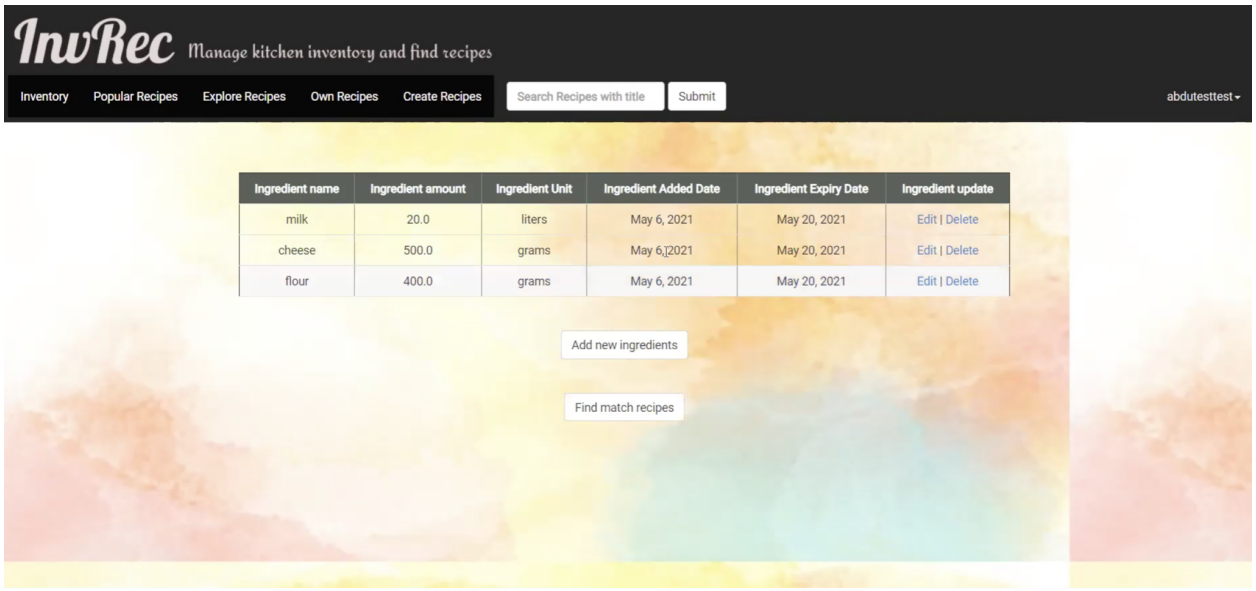


User abudutesttest can go to his own inventory by clicking inventory on navigation bar. Because he is a new user, he will need to create a new inventory or join himself to an existing inventory so that he can share the same inventory as another user.

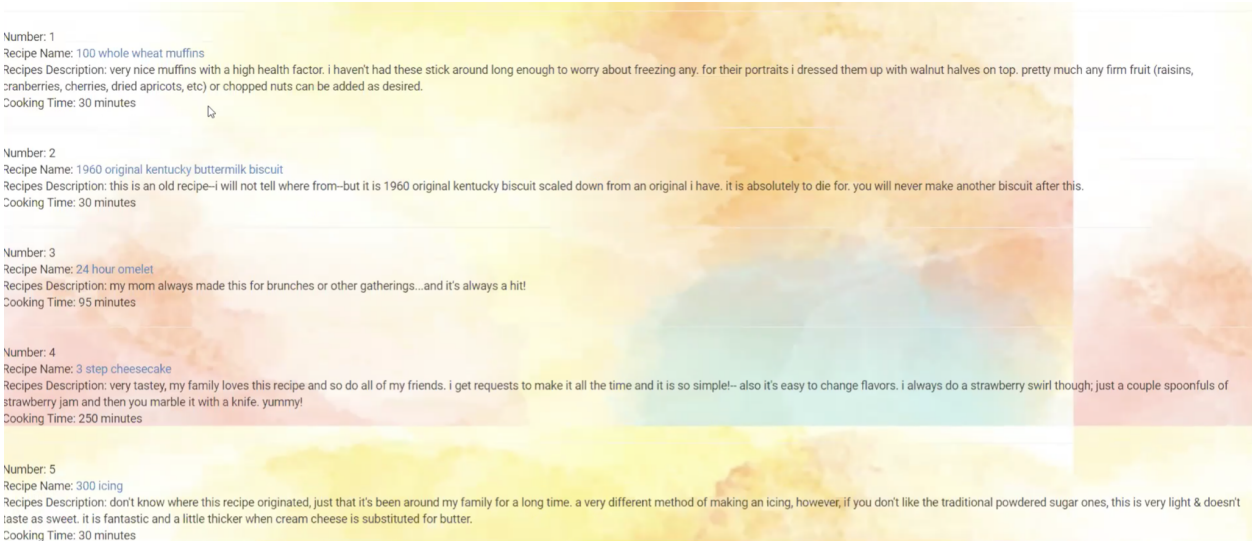


User abdutesttest can then add ingredients, along with the amount of ingredients, units, expiry date to the inventory.

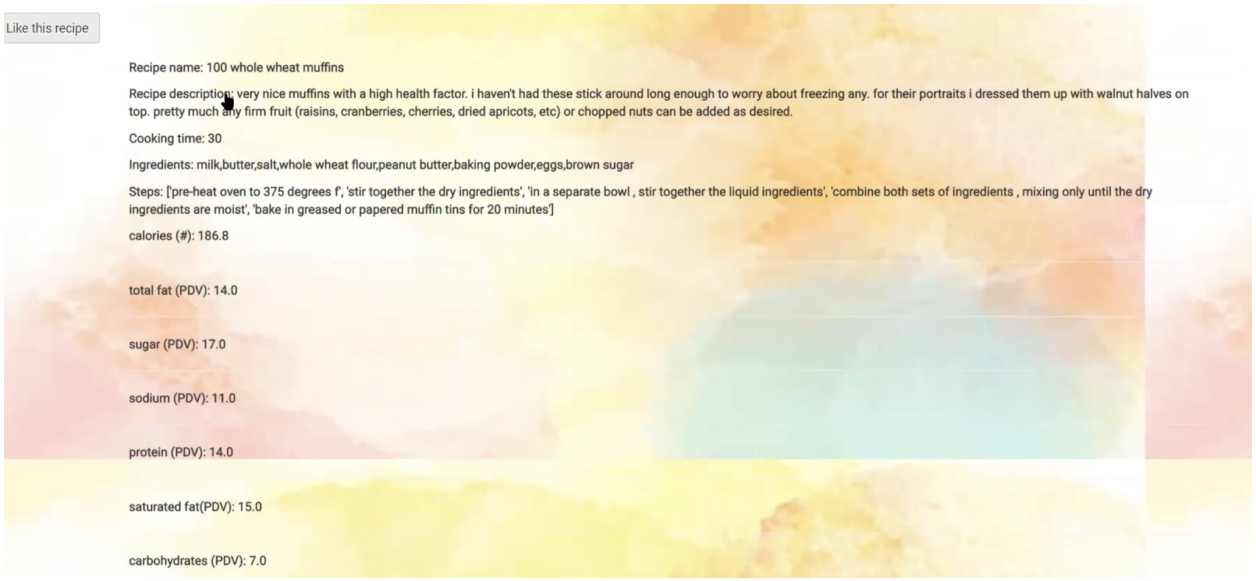




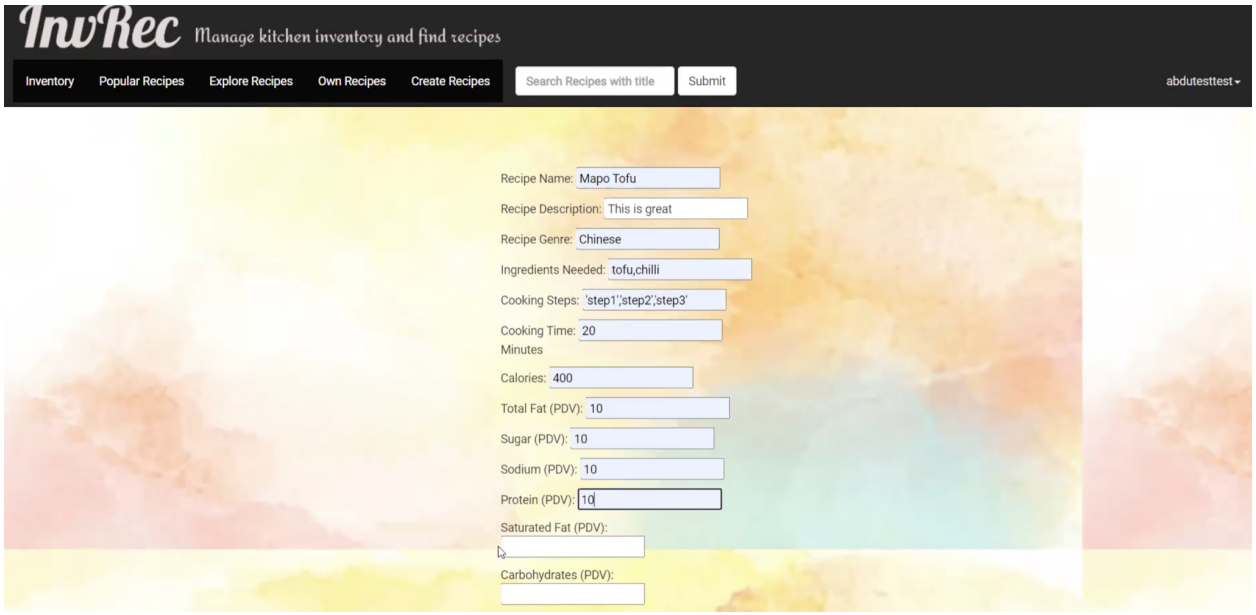
Apart from editing or deleting the ingredients from the inventory, the user can find match recipes based on current ingredients.

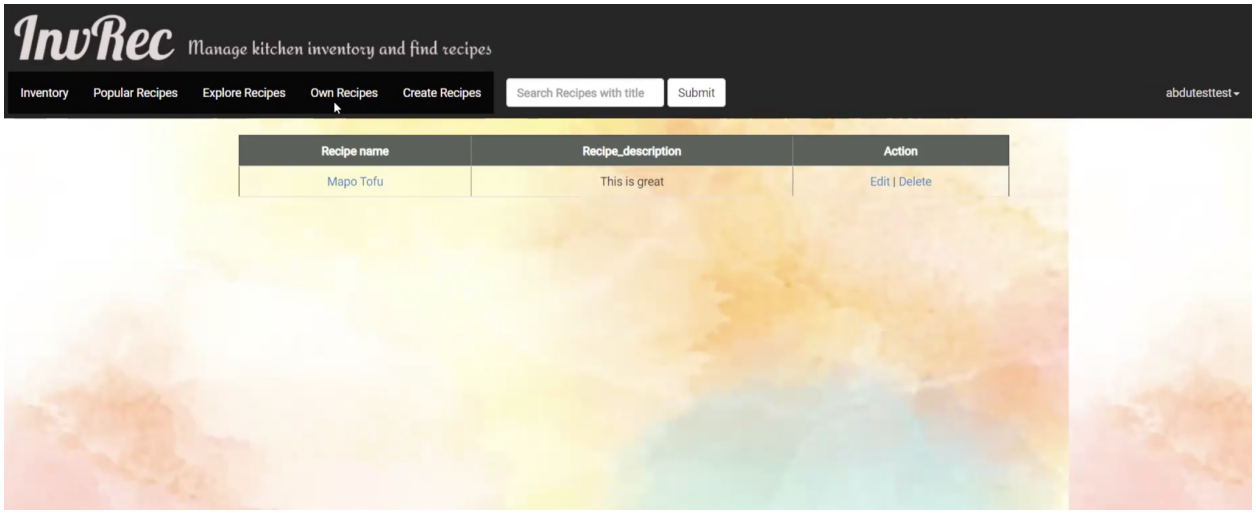


Click on the blue link to view full details of the recipe. The user can see the recipe details, and then proceed to like that recipe.

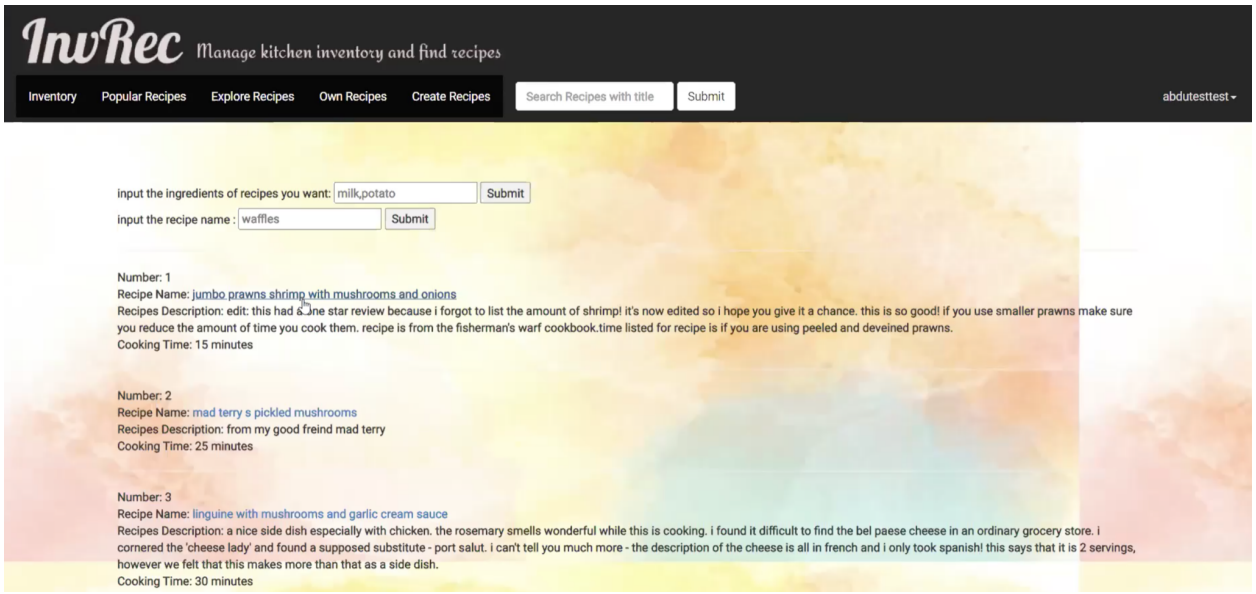


The user can also create his own recipe, and view it in the “Own Recipes” Section

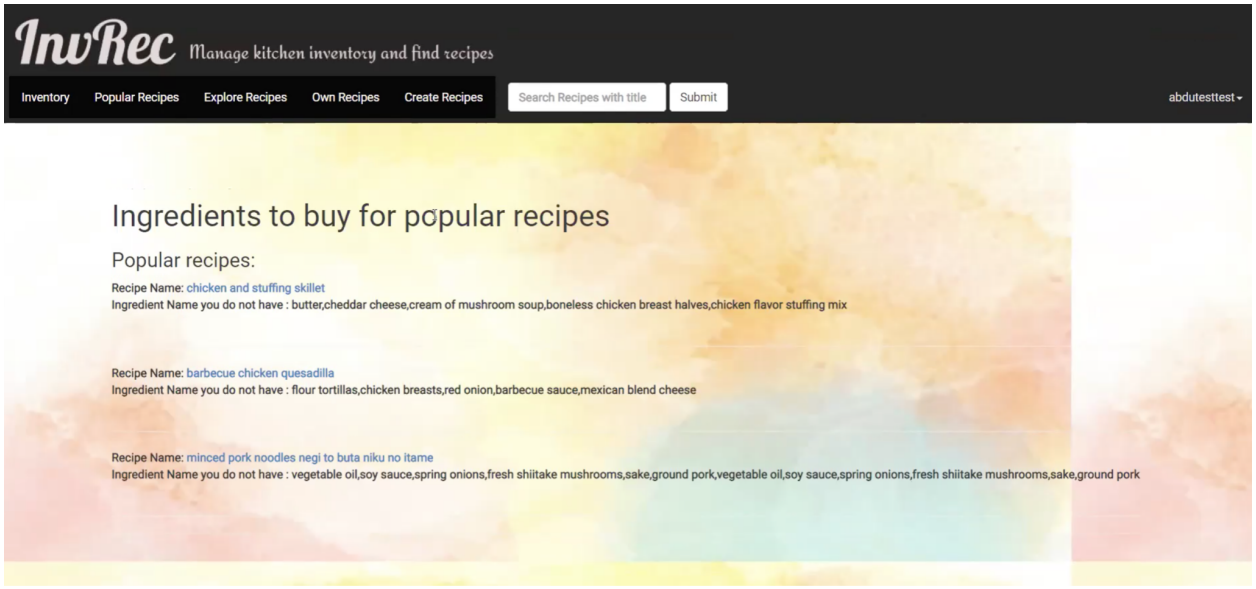




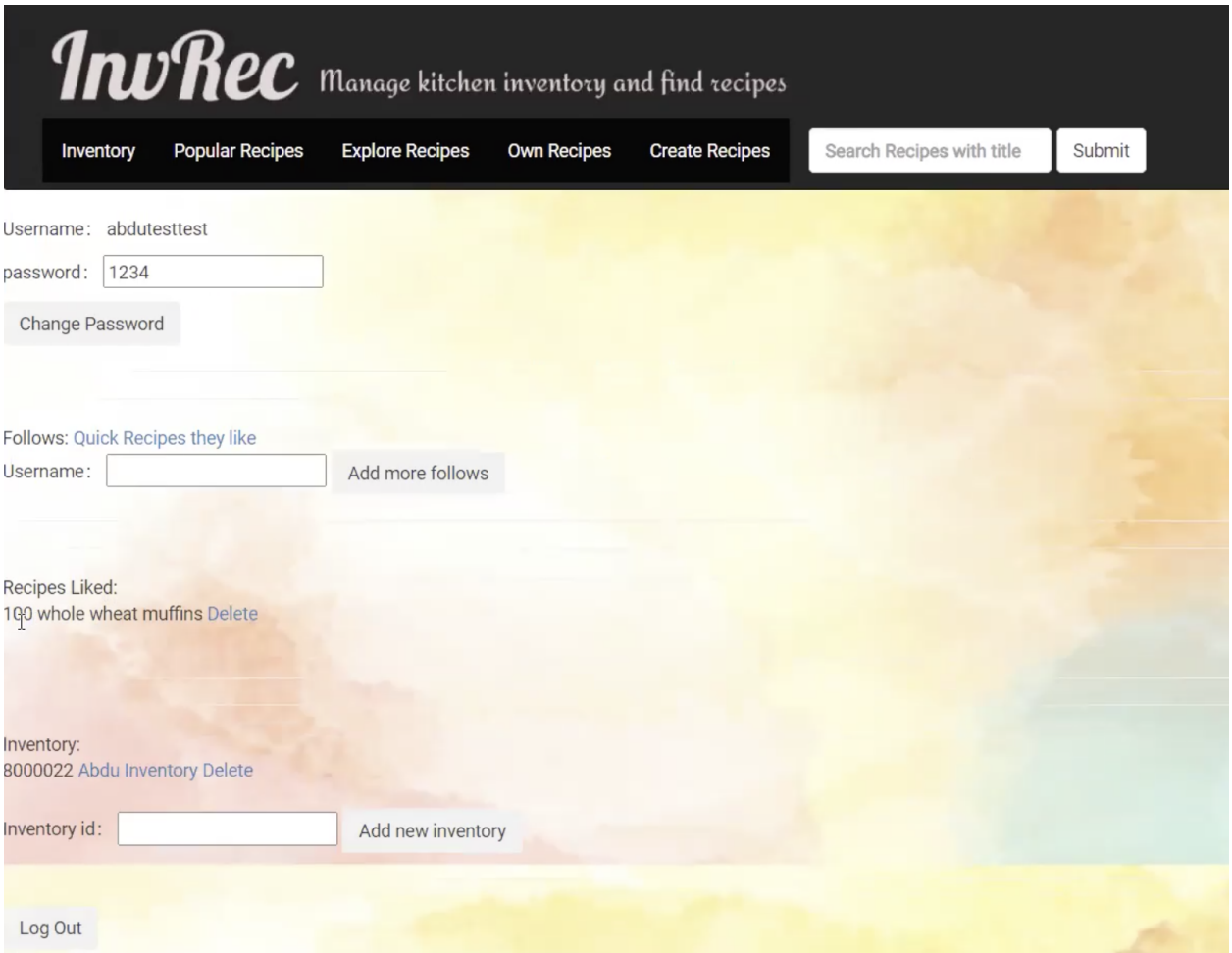
The user can also explore recipes and search recipes based on keywords of recipes or ingredients.



The user can view the popular recipes and get the ingredients missing from his own inventory.

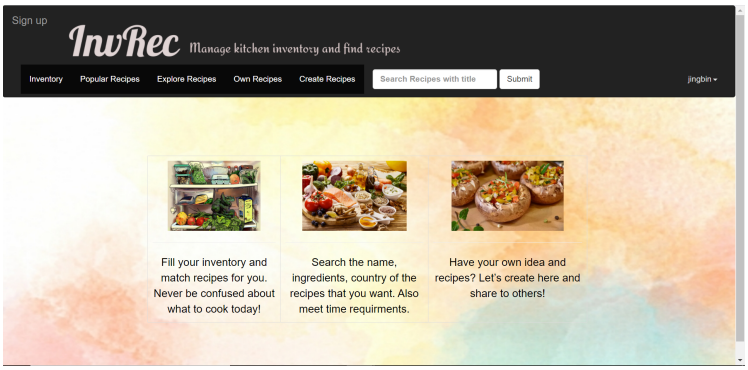


The user abdutesttest can go to his own profile to change the password, follow other users, view all recipes he liked, and manage the inventory he owns.



<u>Technical Challenges:</u>

We encountered a few technical challenges while building our application. One being the difficulty of integrating a UI into our application as none of us had previous UI design experience. A more difficult problem that we encountered is that the django tutorial utilized REST API framework to display their data whereas we wanted to utilize our own custom templates with HTML format to display our data. This required us to look up the method for integrating templates from the django online tutorial and on their official website documentation. The documentation on the django website is extremely comprehensive and it would've been much more efficient for us to read up on those tutorials first as well as the TA guided videos. In addition to this, there are multiple tutorials on youtube regarding creating a navigation bar and it's not completely necessary to create your own web application features completely from scratch.



Evaluation of the implementation process:

For the most part, the implementation process itself went according to plan. We were able to incorporate almost all of the features that we set out to implement. The users are able to access all of the ingredients and all of the different recipes that they would like. The one potential improvement that we could make is by getting the caloric intake and nutritional values for each individual ingredient as opposed to whole recipes. The dataset that we have pulled from makes mention of the nutritional content for the recipes but not the individual ingredients. Thus this one improvement to the implementation process would allow us to calculate the caloric intake for the user's custom made recipes - giving us a more accurate representation of the nutritional value of custom made recipes.

Division of Labor:

Overall our division of labor has generally been pretty efficient. We make use of weekly meetings on Wednesday nights to discuss any of the pressing issues with the project. Having meetings every week, even on the off weeks with no deadlines, has helped us to stay ahead of any assignments that we may have. In our task assignments, we were also very clear with how we wanted to divide up our roles. This was especially prevalent during the data processing. We had a single data source in CSV format however we needed to run different processes to break down the data into individual tables such as ingredients and recipes. As a result, we split this work up relatively easily, fully utilizing the functionalities of python on google collabs.

For the web framework, we also split the work up relatively well. We hosted work sessions of just everyone coding different pages. Kanin was incharge of the ingredients and the

inventory; Gaozheng was incharge of the search and presentation on the recipes table; and Jingbin was incharge of the html templates, users operations, recipes update/insertion and match recipes according ro inventory. Through github we were able to divide the work relatively cleanly and thus allowing us to work very efficiently.