Recipe King

Group #15

Shayne Smither, Jiayu Xia, Kyle Stevenson, Zhiguang Liu, Abdulah Sibalo

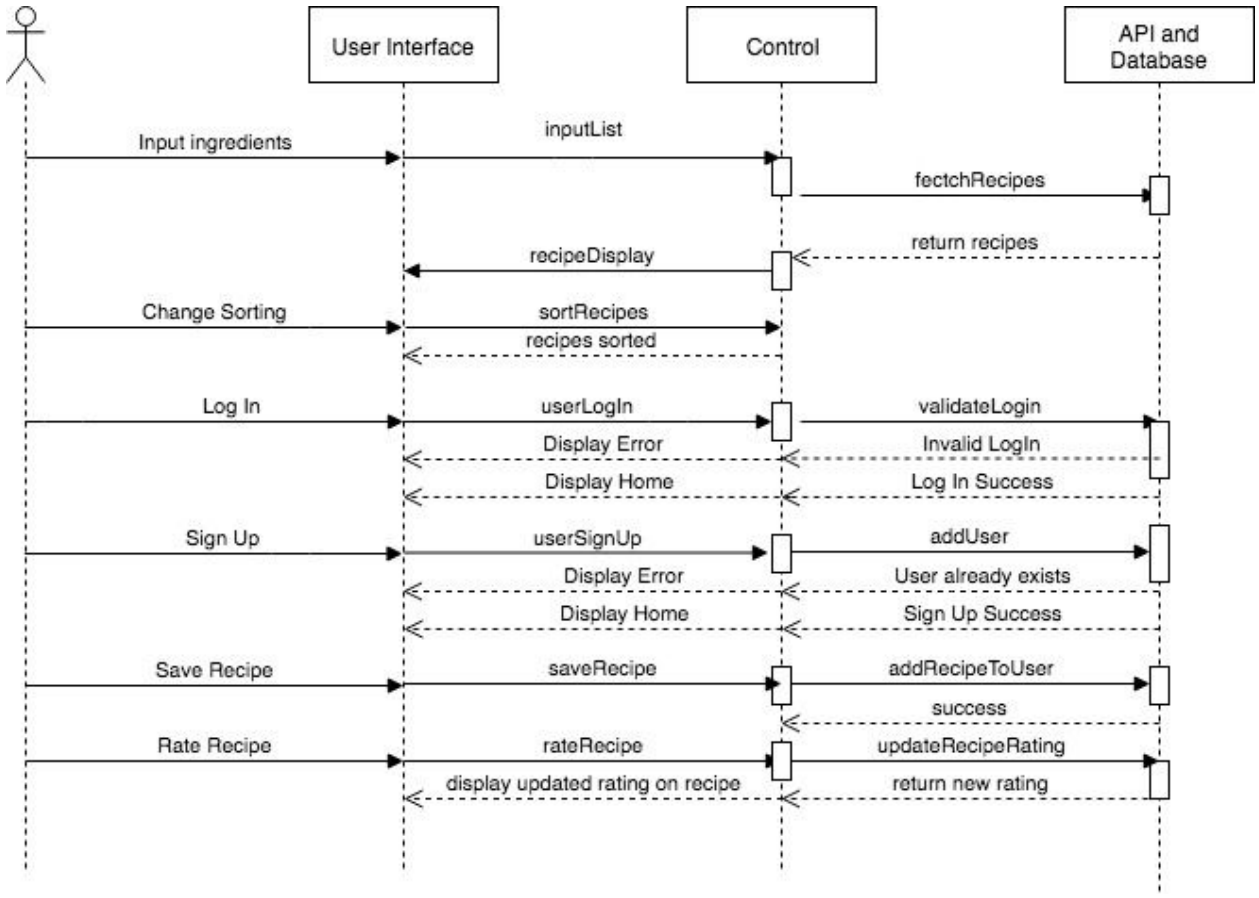
CMP_SC 4320

September 27, 2018

Contents

Report 2

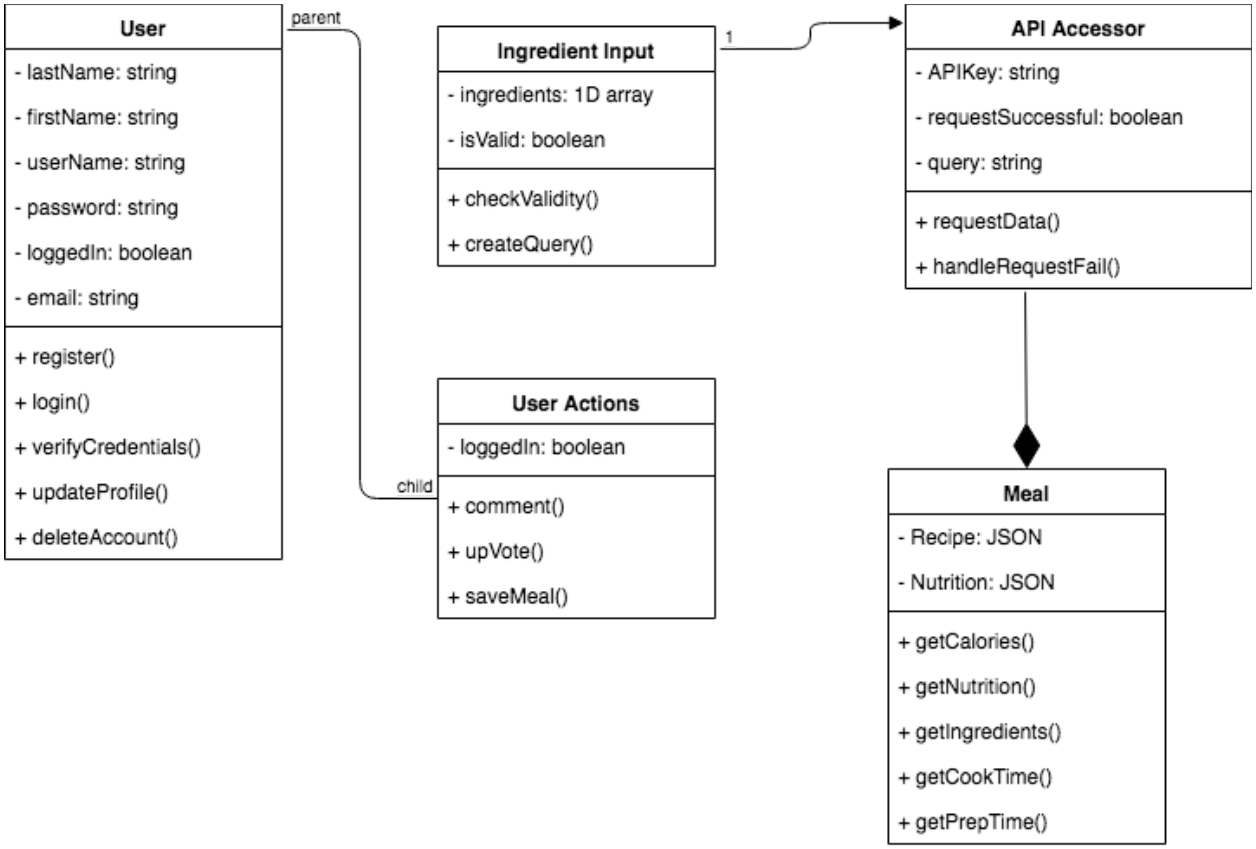
Part 1: Interaction (System Sequence) Diagrams



Part 2: Class Diagrams and Interface Specifications

Class Diagram

The following diagram shows the classes and relationships. While the main purpose of this project will be to provide users a better way of searching for meals by displaying meals for which they have ingredients on-hand, we also include account creation and user action functionality, letting logged-in users comment on, upvote and save meals. The user will enter a list of ingredients, which are then assembled into a query that is sent to the API Accessor. The API Accessor will use the query to request data from the API and handle request failures. If the request is successful, this data is given to the Meal class, which parses the data for calorie and nutritional content, ingredients, cook time, and prep time. The front-end will deal with displaying this data in a clean format, allowing users to sort and filter the data, etc.



Data Types and Operation Signatures

User
Attributes: ● lastName: A string of the user's provided last name. ● firstName: A string of the user's provided first name. ● userName: A string containing the user-specified username for the account. ● password: A string containing the user-specified password for the account. ● loggedIn: A boolen. Used to check if the user is logged in during the session, which then allows the user to comment, upvote, and save recipes. ● email: A string which stores the email address of the user.
Functions

- register(): void. If during the session loggedIn is false, then the user may register for an account. A call to this function will direct the client to the registration page. After the user registers, this function will send the user information to a database.
- login(): void. Allows the user to log in.
- verifyCredentials(): bool. Checks the database for user credentials during log in attempt. Returns true if credentials match user in database, false otherwise.
- updateProfile(): void. Allows the user to update his profile. The user may update any of the attributes associated with his or her account.
- deleteAccount(): void. Allows the user to delete his/her account.

User Actions
Attributes: ● loggedIn: A boolen. Used to check if the user is logged in during the session, which then allows the user to comment, upvote, and save recipes.
Functions ● comment(): void. Allows the user to comment on a recipe if loggedIn is true. ● saveMeal(): void. Allows the user to save a meal if loggedIn is true. ● upVote(): void. Allows the user to upvote a recipe if loggedIn is true.

Ingredient Input

Variables
- ingredients: 1D array of user-specified input.
- isValid: bool.

Functions
- checkValidity(ingredients): bool. Checks the ingredients array to see if every element in the array is a valid character-only string and returns true if it is, false if not. Also sets isValid variable.
- createQuery(isValid, ingredients): void. If the array elements are valid, this function constructs a query to send to the API.

API Accessor

Attributes
- APIKey: strings storing the API key.
- requestSuccessful: bool. True if the request for getting data from the API was successful, false otherwise.
- query: a string storing the query that will be used for getting data from the API.

Functions
- requestData(): void. Submits the request to the API. If successful, sets requestSuccessful variable to true, else to false.
- handleRequestFail(): void. Deals with failed requests by displaying an error message.

Meal
Attributes: ● Recipe: the JSON of the successful API request containing the recipe data. ● Nutrition: the JSON of the successful API request containing the nutrition data about the meals.
Functions: ● getCalories(): int. The calories for the meal. ● getNutrition(): dictionary. A dictionary containing various information about the nutritional information of the meal. The keys are the metric of interest (e.g. carbohydrates) and the values are the corresponding string or integers. ● getIngredients(): array. Returns an array of strings where every element contains an ingredient used in the meal. ● getCookTime(): int. Returns the requisite cook time (in minutes). ● getPrepTime(): int. Returns the requisite preparation time (in minutes).

Traceability Matrix

	Class				
Domain Concepts	User	User Actions	Ingredient Input	API Accessor	Meal
Container for user input			x	x	x
Allow users to create accounts	x	x			
Logged-in user functionality	x	x			
Determine appropriate recipe				x	x
Retrieve recipes from API				x	x

Display meals					x
Sort/filter					

The essential goal of the web application is to optimize the quality of meal recommendations for users based on their available ingredients. The meals displayed after a user inputs the ingredients he has on-hand will be those with ingredients best-matching the user's specification. The concepts were derived based on this criteria, and additional considerations like allowing users to save meals by letting them create accounts.

From the concepts, we derived five classes: user, user actions, API accessor, and meal. The user class is the one that will manages user accounts and stores relevant user data in the database. This class is derived from the desire to allow users to have accounts and save, comment on, and upvote recipes. The child class, user actions, is the class that manages what logged-in users are capable of doing. Essentially, these two classes are for managing users and storing user data to the database.

The ingredient input class checks the input provided on the front end and makes sure the input is valid so that requests to the API are not faulty. The main reason for checking the input is to avoid nonsense calls to the API since the algorithm for making a request to the API will be a compute-heavy algorithm and it would not pay off to allow this algorithm to execute unless the input is valid. The related API Accessor class is where the algorithm for making requests to the API will live. This class is derived from the domain concepts of returning the best-matching recipes from the API. Once we verify that the input is valid, this class will call the algorithm to gather the meals that are (mostly) comprised of ingredients that the user has specified. The purpose of the meal class is then to parse the API data for the relevant information, such a the recipe, caloric data, nutritional facts, cook time, and prep time.

System Architecture and System Design

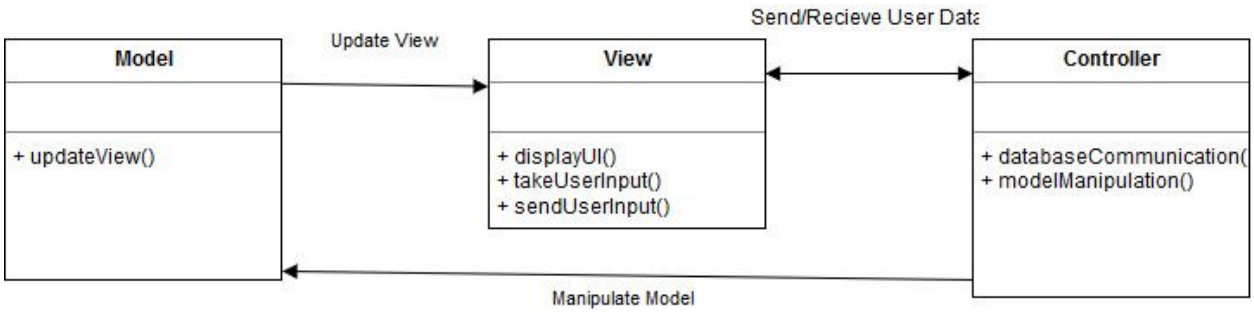
Architectural Style

As our system is a web application, we will be using the Model View Controller (or MVC) architecture pattern since it is a popular choice for web apps and in our case, extremely applicable. The MVC is split into three distinct parts: the Model, the View, and the Controller. The user will directly interact with the View. The View displays the user interface and passes along any user input to the Controller. The Controller then uses the user-input data to manipulate the Model. The Model then updates the View with the new information.

More specific to our system, the user will input their list of ingredients to the View. The View will pass the list to the Controller which will pull the appropriate recipes from a database. The Controller will then use the list of recipes to add data to the Model which will update the View accordingly and show the user their new list of recipes.

Identifying Subsystems

Our system is designed around three primary subsystems. The View will display the UI and be the way our system interfaces with users. The Controller will take the user input and change it into data usable by our system. It will then use that data to generate a list of recipes from a database and manipulate the Model with the newly created list. The Model will use the list to update the user interface and the cycle will begin anew.



Network Protocol

Our system will be a web based application that requires a connection to a third party host server. We will be using HTTPS as our network protocol because many web browsers will throw up warnings when the user attempts to access an HTTP based web page due to concerns over security. Additionally, a connection to a third party database will require an HTTPS to authenticate the API key. Our system will not be storing any information that is a security risk, but for the convenience of the user, as well as utilizing a recipe API, we have decided that HTTPS will be the most appropriate network protocol.

Global Control Flow

Execution Orderness:

The execution order of our system is a mix between procedure-driven and event-driven. That is, the user must always follow the same steps to use the system when starting, but will obtain more agency later on. The user always starts by inputting

ingredients, then will receive back a list of recipes. At this point the user chooses their own event. They can either pick a recipe from the list or sort with their own criteria.

Time Dependency:

Our system will not make use of any timers. It will only act in a reactionary capacity after the user makes their own action.

Hardware Requirements:

- Any device capable of accessing a web page
- A color display to view the webpage
- A third-party web host server

Part 3: Algorithms and Data Structures

Algorithms

In The Recipe king, the main algorithm is searching algorithm. This algorithm is based on our own SQL database of recipes. To understand our algorithm you need to understand the structure of the database first.

The database has two tables, one is User table, another is Recipe table.

The User table has 9 columns:

ID(primary key), lastName, firstName, userName, email, password, numOfSearch, numOfUploads, numOfReviews

The Recipe table has 9 columns:
ID(primary key), providerID, name, ingredients, numOfCal, numOfIngredients, review(int), textreview(text), popularity

In the User Interface, the user will input main indigents he/she want, numOfCal and numOfIngredients are optional.

The searching algorithm is couple groups of SQL statements, for example: a user have potato and tomato as main ingredients;
In case one, the user has only two ingredients, so the user input will be:
main ingredients: potato, tomato
number of indigents: 2
number of Calorie: (default)

The algorithm (sql statements) will be

```
SELECT * FROM Recipe
WHERE indigents LIKE '%potato%'
AND indigents LIKE '%tomato%'
AND numOfIndigents = 2
```

In case two, the user has multiple indigents but he want potato and tomato as the main indigents:
main indigents: potato, tomato
number of indigents: (default)

number of Calorie: (default)

The algorithm (sql statements) will be

SELECT * FROM Recipe
WHERE indigents LIKE ‘%potato%’
AND indigents LIKE ‘%tomato%’

Structure

Because we have our own database so we can directly get everything we want; the structure we used is pretty simple, just an array:

First we need create a query:
$query =
"SELECT * FROM Recipe
WHERE indigents LIKE ‘%potato%’
AND indigents LIKE ‘%tomato%’
AND numOfIndigents = 2";

THen we execute $query, use $result to get the recipe array from database:

$result = $mysqli->query($query);

use a while loop to print a table:

while($row = $result->fetch_assoc()) {

```
print "<tr><td>" . $row['ID'] . "</td><td>" . $row['name'] . "</td><td>" . $row['indigents'] .
"</td><td>" . $row['numOfCal'] . "</td><td>" . $row['review'] . "</td><td>" .
$row['textreview'] . "</td></tr>";
}
$result->close();
$mysqli->close();
```

Part 4: User Interface and Implementation

This section shows a prototype design of the user interface. The first image shows the main page where a user inputs the ingredients they have on-hand and receives a list of possible recipes for available meals. There is also a section to add filters with which to sort the results. The second image shows a recipe's information after it is clicked on. This information includes the recipe itself and associated information (cooking time, nutrition info, etc.), a photo of the completed dish, comments by other users who have made the dish, and a simple rating. The third image shows the login view and once a user log in, one can access to the favourite recipes he stored earlier in order to make the process more efficient which is shown in the fourth image.

Login | Fav

RecipeKing

Filter1

Filter2

Filter3

Search

Recipe	Ingredient	Image
Bacon Avocado Grilled Chicken Sandwich	chicken, bacon, swiss, avocado, sauteed onions, lettuce, tomato, cilantro-pesto mayo	
California Turkey Club	Bacon, avocado, tomato, red onion, swiss, lettuce, cilantro-pesto mayo	
Buffalo Chicken Ranch Sandwich	chicken, spicy Buffalo sauce, tomato, lettuce, house-made ranch	
Spicy Shrimp Tacos	pico, avocado, coleslaw, queso fresco, Mexican rice, black beans	
Bacon Ranch Chicken Quesadillas	Chicken, shredded cheese, chile spices, bacon, house-made ranch, pico, sour cream, ancho-chile ranch.	

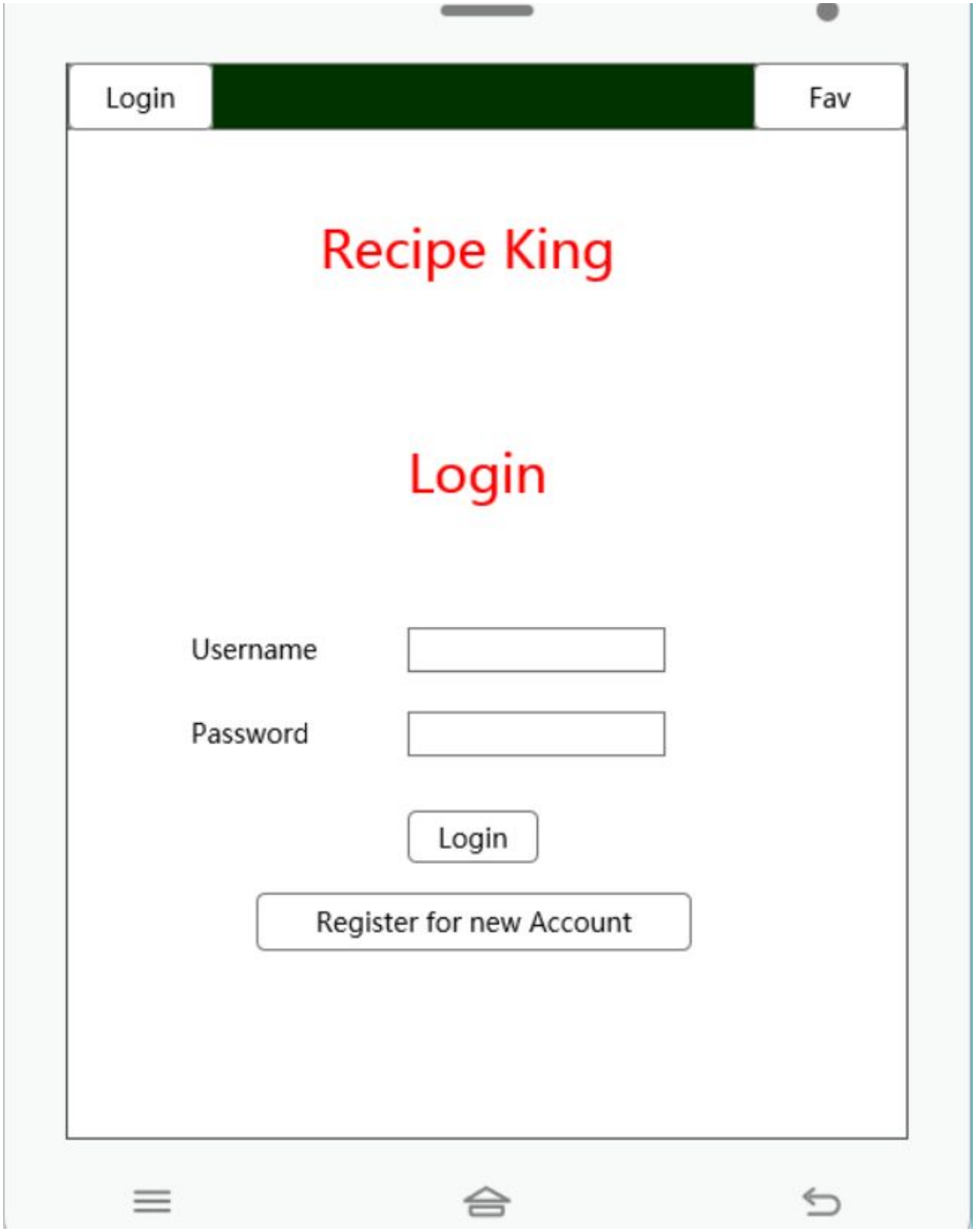
Login

Fav

Bacon Avocado Grilled Chicken Sandwich

Set as Favourite

Etiam nisi lorem, posuere at turpis at, fringilla efficitur quam. Suspendisse vitae lacus ac lectus facilisis ornare. Vivamus vitae pulvinar nisi, in vehicula elit. Praesent iaculis ante tellus, eu mattis lectus suscipit sit amet. Sed congue accumsan nunc in iaculis. Sed malesuada elit turpis, eu egestas eros rhoncus non. Sed pulvinar euismod libero sit amet scelerisque. Vestibulum ante felis, condimentum in vulputate id, tempor eu nulla. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis


Login
Fav
Recipe King
Login
Username
Password
Login
Register for new Account

Part 5: Design of Tests

A.Test Case

Use Case UC-1: inputList
Related Requirements: REQ-1 **Initiating Actor:** User Interface **Participating Actor:** Recipe Fetcher **Actor's Goal:** Input a list of ingredients that will be used to generate a corresponding list of recipes. **Preconditions:** ● The web page has loaded correctly with all UI elements displaying properly **Postconditions:** ● The system returns a list of recipes corresponding to the ingredients input by the user **Flow of Events:** ● User inputs the ingredients which are fed into inputList(); ● inputList() is passed to fetchRecipes(), which makes a request to the API to receive the recipes based on the ingredients provided.

Use Case UC-2: fetchRecipes

Related Requirements: REQ-1

Initiating Actor: Recipe Fetcher

Participating Actor: User Interface, API

Actor's Goal: Generate a list of recipes from a database based off of a provided list of ingredients.

Preconditions:

- A list of ingredients is provided

Postconditions:

- The system returns a list of recipes corresponding to the ingredients input by the user

Flow of Events:

- A list of recipes is returned to the web browser based on what was contained in inputList().
- The browser display the available recipes along with photos of the prepared meals.

Use Case UC-4: profileManagement

Related Requirements: REQ-3, REQ-4, REQ-5, REQ-6, REQ-8

Initiating Actor: User

Participating Actors: User Interface, User Database

Actor's Goal: Create and display data based on the logged in user

Preconditions:

- A user is logged in.

Postconditions:

- The system displays data that the user has previously created such as comments, ratings, and saved recipes

Flow of Events for Main Success Scenario (User is logged in):

- The user comments on the recipe
- The user's comment is saved in the database

  OR
- The user favorites/saves the recipe
- The data is saved in the database

  OR
- The user rates the recipe
- The rating information is added to the total in the database and the average user rating is updated and displayed

Flow of Events for Alternate Success Scenario (User is not logged in):

- The user selects the recipe he/she wants.
- No data is saved/stored, and the user cannot comment on, favorite, or rate a recipe.

B. Test Coverage

The tests cover most essential classes implemented and some useless as well as synchronization of the database to the application.

More tests will be implement if there are more low-priority features are available.

C. Integration Testing Strategy

We are using Big Bang Approach integration testing strategy.

Our system is not a rather complicated one and Big Bang Approach saves us time to find the whole system is worked well or not. Once if an error showed, we can easily locate the bug since every module is working independently and we don't need to worry about the situations they may disturb and loop with each other in any case. The re-test after fixing will be time-consuming as well.

Part 6: Project Management and Plan of Work

Merging and Collecting Contributions:

All information is first gathered on a google doc and each member contributes his part of the project onto the google doc. A team member then takes all information and forms a PDF separately to keep consistent with the rest.

Plan of Work:

Our team will be divided into two teams. One team will work on the front-end of the website. The second team will work on the back-end.The front-end team will be Kyle Stevenson and Jiayu Xia. The back-end team will be Abdulah Sibalo, Shayne Smither, and Zhiguang Liu. We will use the VOIP program Discord to communicate as well as in-person meetings on an as-needed basis.

The front-end team will work on the following tasks:

- Building the input form for the recipe search.
- Display of recipe results to the user.
- Allow user filtering and sorting of results.
- Develop a consistent user experience across all devices.

- Add additional search options.
- Develop front-end functionality of low-priority features:
    - User login and sign up page
    - Rating, Commenting, and Saving recipes
    - User uploaded images to recipes

The back-end team will work on the following tasks:

- Server side functionality to call api service, and return results to the front-end.
- Develop back-end functionality of low-priority features:
    - Create a database and schema to track users.
    - Save user data related to recipes in database: Saved recipes, Comments, Ratings.
    - Allow for images to be saved for related recipes.

Current Status:

We are currently finishing up report 2, and getting everything ready for the demo presentation. We also have the first draft of the website working online by now.

Future Status:

After the demo presentation, we will join our efforts to the website building in order to display the results of the recipes though API and our own database.

Part 7: References

1.UMLDesign: gliffy.com

2.Moqups: https://app.moqups.com/

3.Sun Microsystems: http://java.sun.com/products/jlf/ed2/book/

4.Wikipedia Definition of User Stories: http://en.wikipedia.org/wiki/User_story

5.The Software Engineering textbook by Ivan Marsic. Link at: http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.p