Recipe King

Group #15

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

CMP_SC 4320

November 30, 2018

Contents

Summary of Changes

- Additional requirement (REQ-9) added to functional requirements
- Additional use cases added
- Updates to Traceability matrix
- Updates to diagrams to reflect current design
- Updates to on-screen appearance requirements
- Updates to test cases and TC-1,2,3,4 added
- Updates to current status

Customer Statement of Requirements

Problem Statement

Preparing a meal does not come naturally to most. For many people, the process of preparing a meal is quite involved, and there is a steep learning curve to cooking and preparing satisfying meals. The biggest issue faced by most is the issue of looking for and finding a meal worth preparing that fits within their constraints. We are constrained by time, budget, knowledge, materials, etc. Time and materials available are an especially relevant constraint when we are considering what to prepare in the moment. There are hundreds of thousands of meals available, so a great deal of time can be wasted searching for something. Additionally, when you find something that interests you, the cruel reality may be that you simply don't have nearly enough ingredients to prepare the meal as it is specified.

All these problems compound when you factor in health. A trip to any of the myriad of fast-food places is tempting when you don't have time to prepare a meal for yourself, sometimes due to your limited awareness of the meals that can be prepared with the ingredients you have on-hand. Yet eating at any restaurant, fast-food or not, is also considerably more expensive than preparing your own food in terms of real monetary expenses and physical health costs. In short, it is difficult to do your own cooking due to the constraints of individual creativity, time, and effort required, yet it is desirable to do so since the alternatives are often unhealthy and considerably more expensive.

These are all problems that cost people a lot of time, effort, and money. We are busy and we value convenience, this much is clear. In order to bypass the hassle and time-cost of cooking, we turn to convenient alternatives like fast-food, as previously mentioned. Similarly, we settle for frozen meals and resort to over-relying on our

microwaves. These alternatives, however, are unhealthy and more expensive in the long run.

Also indicative that there is a real problem is the recent rise of meal-kit delivery companies, which promise to deliver fresh ingredients for a pre-planned meal to one's door. The main value proposition of such companies is that you don't have to spend hours looking for meals and then going to the grocery store to buy the requisite materials. Instead, the ingredients are nicely packaged for you along with an easy-to-follow recipe guide. The issue with this solution is that the meal-kits often cost significantly more than if you were to buy the ingredients yourself, and do not always arrive in perfect condition, if at all, due to the sensitive and vulnerable supply chain of such companies.

Current treatments are inadequate; cookbooks and online platforms do not deliver a customizable service to the client. Indeed, these solutions typically just list meals that are available in the database, their ingredients, and how to prepare the meal. A user may have to spend significant time searching for something worth devoting their time to preparing. The typical process goes something along the lines of: look for a meal out of the countless many offered, read through the ingredients only to realize you don't have the materials necessary, and continue to iterate through this process until you find something suitable. There is significant time lost and it is a pain to find a suitable meal.

Proposed Solution

We need a solution that emphasizes convenience and is capable of catering to the masses who want to cook but are constrained by various factors, and this is what we are proposing to build. Our goal is to build a convenient and easy-to-use web application that can offer a simple and convenient solution to the problems discussed above.

System Requirements

Enumerated Functional Requirements

Below are the functional requirements of the web application. These were created with the customer in mind, and we believe these are the components necessary to meet our proposed solution criteria and create a satisfactory experience. We follow the convention of using the phrasing “the system shall” for high-priority and necessary requirements and “the system should” for lower-priority items. The priority weights range from 1 to 5, with 5 being highest priority and 1 being lowest.

Identifier	Priority	Description
REQ-1	5	The system shall take a list of ingredients as input and give a list of recipes sourced from an API as output.
REQ-2	4	The system shall function on multiple devices such as computers and mobile devices.
REQ-3	3	The system should allow users to create a profile.
REQ-4	3	The system should be able to store user-provided recipes.
REQ-5	2	The system should allow users to comment on recipes.
REQ-6	2	The system should allow users to give recipes a rating and save recipes.
REQ-7	4	The system shall allow users to sort and filter

		recipes based on various criteria.
REQ-8	1	The system should allow users to upload images.
REQ-9	1	The system should check user-entered ingredients against a list of valid ingredients to make sure results are returned from the API call.

These functional requirements specify the end-goals of the system. The value-add of our web application is two-fold: first, we propose to give users the ability to enter multiple ingredients at once and have all of them be contained within the returned meals. Second, the application will allow the user additional freedom to sort and filter based on various criteria (caloric content, kosher/halal, etc.).

Enumerated Non-functional Requirements

The following are the non-functional requirements. We follow the same conventions as with the functional requirements.

Identifier	Priority	Description
REQ-9	5	The system shall be easily usable and not require too much time to process input or display results client-side.
REQ-10	3	The system should be maintainable and not rely exclusively on the API for providing the user with recipe data.
REQ-11	2	The system should be scalable to accommodate a growing user base.

REQ-12	1	The system should be able to handle thousands of requests at once.

On-Screen Appearance Requirements

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. The resulting recipes are output into a table and can be sorted by the user alphabetically by recipe name, by number of ingredients, or by calories. The second image shows a recipe's information after it is clicked on. This information includes the recipe itself and associated information including an image of the dish, full ingredient list, serving size, and calorie count. The web app is also responsive and mobile friendly to provide a good user interface regardless of device.

Recipe King Home About

Recipe Search

Add your list of ingredients below

Ingredient #1 x

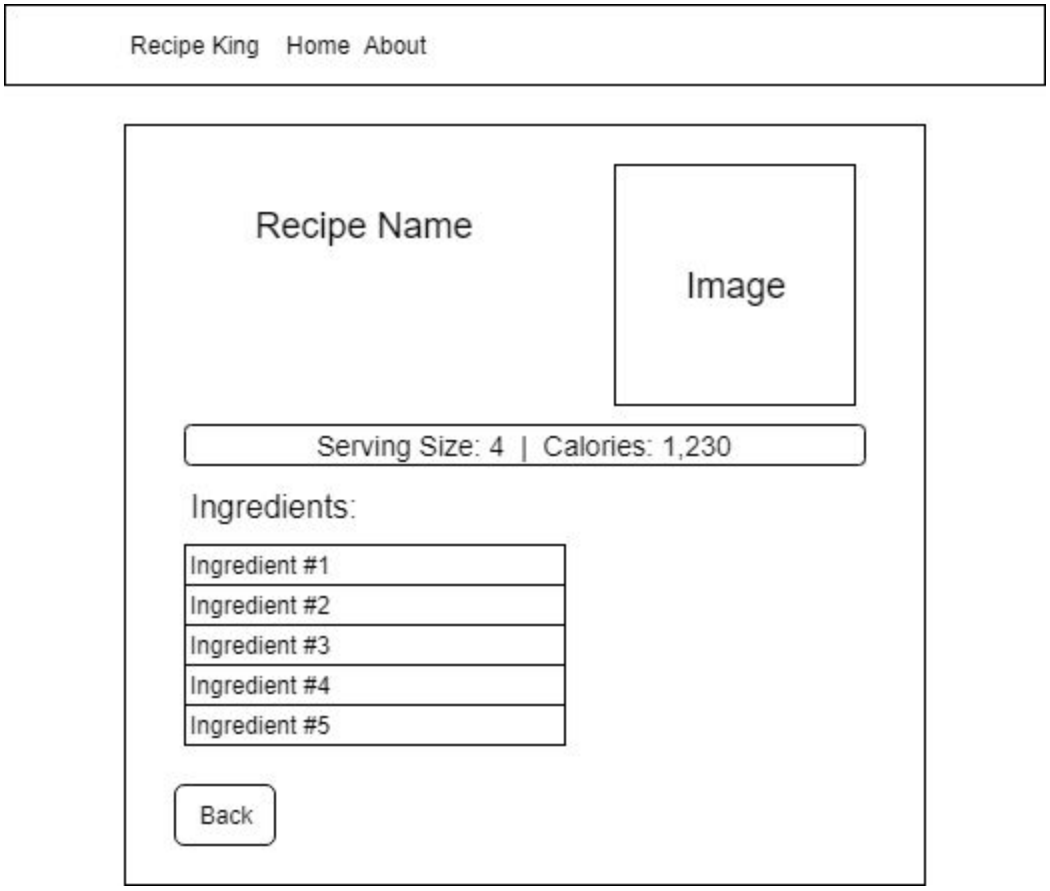
Ingredient #2 x

Ingredient #3 x

Add

Search Recipes

Recipe		# Ingredients	Servings	Calories
Image	Recipe #1 Name	4	3	541
Image	Recipe #2 Name	8	4	862
Image	Recipe #3 Name	15	5	1,254



Functional Requirements Specification

Stakeholders

Stakeholders are anyone and everyone who has a vested interest in a product or system. The amount of stakeholders attached to any given system largely depends on the versatility and nature of the system itself. In the case of Recipe King, stakeholders include anyone who has the desire or responsibility to put together food for themselves and/or others. The vast majority of people on Earth are at least semi-responsible for feeding themselves or others and are therefore potential stakeholders in Recipe King. However, specific examples of potential stakeholders include:

1. College students who do not live on campus and do not have a dining plan with access to cafeterias;
2. Working adults who have full-time jobs spending most of their daylight hours away from home and their kitchen. They likely have little time to grocery shop let alone formulate meal ideas and execute them;

3. Parents, guardians and any form of caretakers who must provide meals for children or others that can not provide for themselves;
4. High school students who require or desire more freedom and responsibility for what they consume.

Actors and Goals

There are few actors in the case of a simple web application. The user initiates all behavior, and the API will participate in retrieving data. In addition, there is of course the user interface and the database.

Actor	Actor's Goal
User (initiator)	Receive a list of possible meals that can be prepared with ingredients on-hand
User (initiator)	Choose a certain recipe and see the details of how to create it, what is required, and an image of the completed recipe
User (initiator)	Create a profile with which to save, comment on, and rate recipes
API (participator)	Retrieve data for the user as-needed.
User Interface (participator)	Load data elements and display for the user.
Database (participator)	CRUD operations as needed.

The user will interact with the client, entering ingredients which will then be sent to the server for processing via API calls. The API will retrieve the data and display it client-side, and then the user will be able to view this data.

Use Cases

The web application will serve primarily to benefit users looking to prepare meals in little time and with ingredients on-hand. With this goal in mind, several use cases may be derived to fulfill the user's needs, and these are described below.

Casual Description

Use Case	Action	Description
UC-1	inputList	The user can enter a list of their currently owned ingredients.
UC-2	fetchRecipes	The system will retrieve possible recipes based on the user-specified ingredients.
UC-3	recipeFilter	The system filters the recipes based on user-specified criteria.
UC-4	profileManagement	The user creates a profile in the system to store data.
UC-5	recipeSort	The system sorts the recipes based off of user criteria.
UC-6	comment	The user can comment on a recipe if logged into a profile.
UC-7	save	The user can save a recipe if logged into a profile.
UC-8	rate	The user can rate a recipe if logged into a profile.
UC-9	recipeCreate	The user can create a recipe to be added to the database.

Traceability Matrix

	Priority Weight	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9
REQ-1	5	X	X							
REQ-2	4	X	X	X	X	X	X	X	X	X
REQ-3	3				X					X
REQ-4	3				X					
REQ-5	2				X		X			
REQ-6	2				X			X	X	
REQ-7	4			X		X				
REQ-8	1				X					
Total Weight		9	9	8	13	8	6	6	6	7

Fully-Dressed Description of Use Cases

Use Case UC-1: inputList

Related Requirements: REQ-1, REQ-2

Initiating Actor: User

Participating Actor: User Interface

Actor's Goal: Input a list of ingredients that will be used to send a request to the API.

Preconditions:

- The web page has loaded correctly with all UI elements displaying properly.

Postconditions:

- The list of recipes is stored in a query which will later be passed to fetchRecipes().

Flow of Events for Main Success Scenario:

- User inputs the ingredients which are fed into inputList();
- The ingredients are valid and all of them should be contained in the recipe to a meal;
- 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, REQ-2

Initiating Actor: User

Participating Actor: API, User Interface

Actor's Goal: Gather a list of meals and associated meal data from a database based off of a provided list of ingredients.

Preconditions:

- A list of ingredients is provided;
- UI elements load correctly;
- Ingredients provided by the user were a valid set of ingredients contained within at least one existing recipe in the database.

Postconditions:

- The system returns meals and associated meal data corresponding to the ingredients input by the user

Flow of Events for Main Success Scenario:

- A list of meals is returned to the web browser based on what was contained in inputList().
- The browser display the meals along with photos of them prepared.
- Each meal has associated meal data which can be accessed by clicking on the image or the title of the meal.

Use Case UC-3: recipeFilter

Related Requirements: REQ-7, REQ-2

Initiating Actor: User

Participating Actor: User Interface

Actor's Goal: Filter the meals by whether or not the recipes are kosher/halal, the caloric range, and diet labels (e.g. "high protein" or "low carb").

Preconditions:

- A list of ingredients is provided;
- UI elements load correctly;
- Ingredients provided by the user were a valid set of ingredients contained within at least one existing recipe in the database.
- The user must specify the filters before calling requesting the meal.

Postconditions:

- The system returns meals and associated meal data corresponding to the ingredients input by the user and the filters applied.

Flow of Events for Main Success Scenario:

- Input list of ingredients
- Apply filters
- A list of meals is returned to the web browser based on what was contained in inputList() and the filters that were applied to the search.
- The browser display the meals along with photos of them prepared.
- Each meal has associated meal data which can be accessed by clicking on the image or the title of the meal.

Flow of Events for Alternate Success Scenario:

- Input list of ingredients
- A list of meals is returned to the web browser based on what was contained in inputList()
- The browser display the meals along with photos of them prepared.
- Each meal has associated meal data which can be accessed by clicking on the image or the title of the meal.

Use Case UC-4: profileManagement

Related Requirements: REQ-2, REQ-3, REQ-4, REQ-5, REQ-6, REQ-8

Initiating Actor: User

Participating Actors: User Interface, 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.

Use Case UC-5: recipeSort

Related Requirements: REQ-7, REQ-2

Initiating Actor: User

Participating Actor: User Interface

Actor's Goal: Sort the meals by calories, number of servings, and/or number of ingredients.

Preconditions:

- A list of meals was retrieved from the user-specified search criteria.

Postconditions:

- The user can sort the recipes by calories, number of ingredients, and number of servings by clicking on the corresponding table data.

Flow of Events for Main Success Scenario:

- The browser display the meals along with photos of them prepared.
- Each meal has associated meal data which can be accessed by clicking on the image or the title of the meal.
- Click on the table data corresponding to the sorting element of interest and the rows will be sorted in ascending order.

Flow of Events for Alternate Success Scenario:

- The browser display the meals along with photos of them prepared.
- Each meal has associated meal data which can be accessed by clicking on the image or the title of the meal.

Use Case UC-6: comment

Related Requirements: REQ-2, REQ-5

Initiating Actor: User

Participating Actor: User Interface, Database

Actor's Goal: Leave a comment on a recipe of choice.

Preconditions:

- The user is on the webpage of the recipe on which he/she wants to post a

comment.

Postconditions:

- The comment is left of the webpage beneath the meal and stored to the database as a record of comments left by the user.

Flow of Events for Main Success Scenario:

- Meals have been returned and are displayed on the UI.
- The user follows the link to the recipe page of interest.
- The user clicks to comment, types his/her comment, and posts it.

Use Case UC-7: save

Related Requirements: REQ-2, REQ-6

Initiating Actor: User

Participating Actor: User Interface, Database

Actor's Goal: Save a recipe of choice.

Preconditions:

- The user is on the webpage of the recipe which he/she intends to save for reference.

Postconditions:

- The recipe is stored to the user's profile, which can be referenced at any time, and in the database.

Flow of Events for Main Success Scenario:

- Meals have been returned and are displayed on the UI.
- The user follows the link to the recipe page of interest.
- The user clicks to save the recipe to his/her profile.

Use Case UC-6: rate
Related Requirements: REQ-2, REQ-6 **Initiating Actor:** User **Participating Actor:** User Interface, Database **Actor's Goal:** Leave a rating on a recipe of choice. **Preconditions:** ● The user is on the webpage of the recipe which he/she desires to rate. **Postconditions:** ● The rating is averaged with the others and displayed next to the image of the meal. ● The rating is stored in the database. **Flow of Events for Main Success Scenario:** ● Meals have been returned and are displayed on the UI. ● The user follows the link to the recipe page of interest. ● The user clicks to rate, rates from 1-5, and submits the rating.

Effort Estimation

In order to quantify the effort it will take to complete this project, we need to calculate the Use Case Points (UCP).

$$UCP=UUCP*TCF*ECF$$

Unadjusted Use Case Points (UUCP) are found by adding two components:

1. The Unadjusted Actor Weight (UAW).
2. The Unadjusted Use Case Weight (UUCW).

UAW is calculated by adding the complexities of all the actors in the use cases.

UAW=Users(3)+API(1)
UAW=4

UUCW is calculated by adding together the weights found in the traceability matrix.

UUCW=9+9+8+13+8+6+6+6+7=72

Therefore:

UUCP=72+4=76

TCF (Technical Complexity Factor) is found with the equation: TCF=C1+C2*TFT
In order to find TFT (Technical Factor Total) we look at the table below.

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	Easy to use efficient interface	1.5	3	4.5
T2	Not rely exclusively on API for recipe data	1	4	4
T3	Scalable to accommodate as many users as necessary	1	2	2
T4	Ability to process thousands of requests simultaneously	2	2	4
	Technical Factor Total (TFT)			14.5

Since C1=0.6 and C2=0.01,

TCF=0.6+(0.01*14.5)=0.745

ECF (Environment Complexity Factor) is found by quantifying the experience level of the team members as well as the stability of the project. It is calculated using information in the table below.

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor
E1	Mostly beginners at UML based development	1.5	1	1.5
E2	Intermediate familiarity with application problem	.5	2.5	1.25
E3	Mostly beginners at the Object-Oriented approach	1.5	2	3
E4	Somewhat motivated about the problem	1	4	4
E5	Programming language proficiency	2	4	8
	Environmental Factor Total (EFT)			17.75

The equation to calculate ECF is ECF=C1+C2*EFT where C1=1.4 and C2=-0.03. Therefore:

$$ECF=1.4+(-0.03*17.75)=0.8675$$

We calculate the final UCP to be:

$$UCP=76*0.745*0.8675=49.12$$

If we assume that the Productivity Factor is 20 hours per user case point, the effort estimation would be (49.12*20), or 982.4.

Domain Analysis

Domain Model - Concept Definitions

Responsibility	Type	Concept Name
Coordinates all interactions between UI and inputs/requests, controls API access and data retrieval	D	Controller
Displaying UI in browser	D	Viewer
Container for user's input ingredient list	K	IngredientStorage
Container for recipes available and their ingredients and associated data	K	RecipeStorage
Checking ingredients against those in database	D	IngredientChecker
Determining appropriate recipes for user	D	RecipeMatcher
Retrieving recipes from API based on input ingredients	D	RecipeRetriever
Updating UI to show recipes and associated data	D	Viewer
Allowing user to sort and filter based on preferences	D	ViewCustomizer
Allowing user to create an account and log in/out	D	AccountCreator
Container for user's profile data	K	AccountStorage

Logging activity of logged-in users (comments, favorites, and recipes ratings)	D	Logger

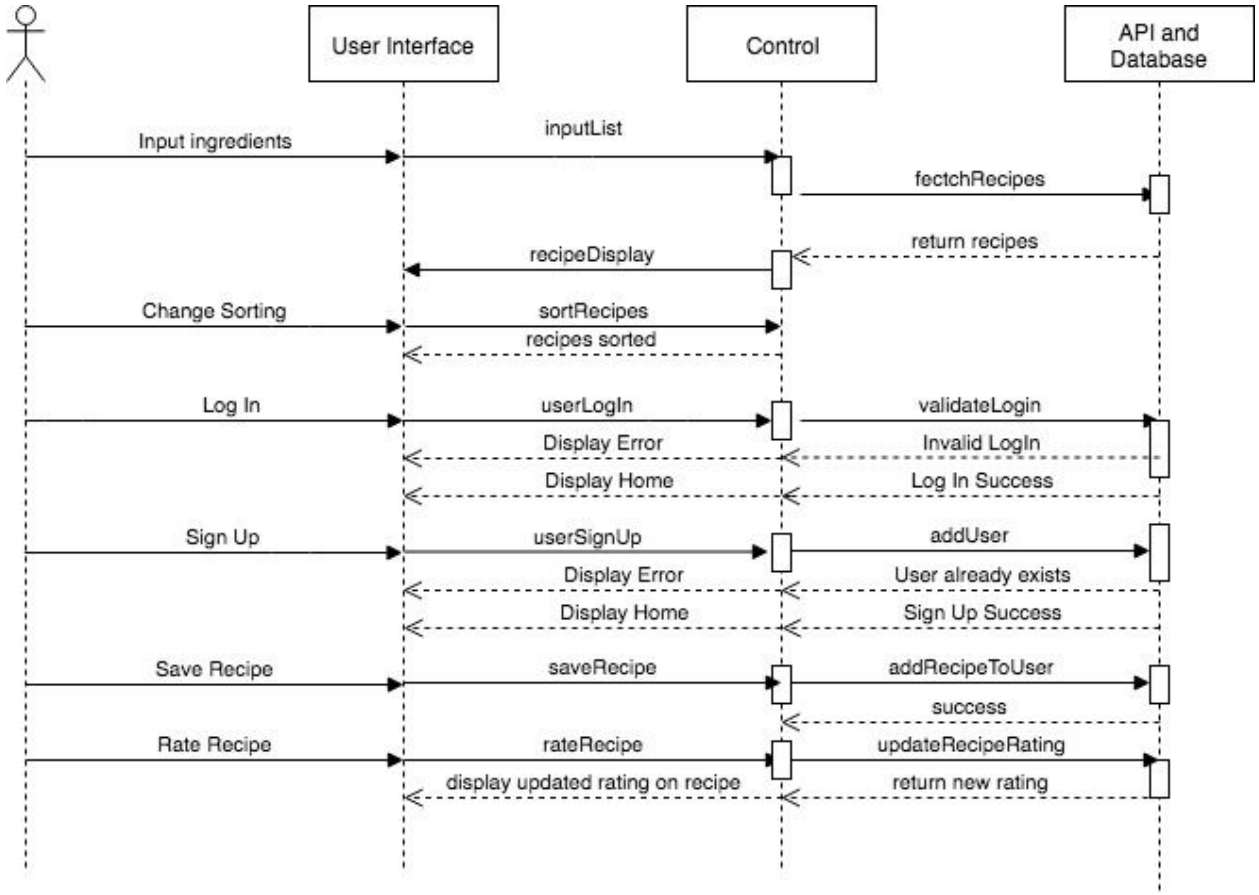
Association Definitions

Concept Pair	Association Description	Association Name
Controller ↔ Viewer	Controller passes requests to the Viewer and receives updated webpage	Conveys requests
RecipeRetriever → Viewer	Data from RecipeRetriever passed to View to display	Displays recipes
ViewCustomizer → Viewer	Filters and/or sorting request sent to Viewer to update the displayer recipes	Update displayed recipes

Attribute Definitions

Concept	Attributes	Attribute Description
IngredientChecker	IsMajority	Check for recipes with at least some threshold (TBD) of user-specified ingredients.
ViewCustomizer	Ascending, Descending, CookTime, PrepTime	Sorts or filters the View based on user-specified requirements.

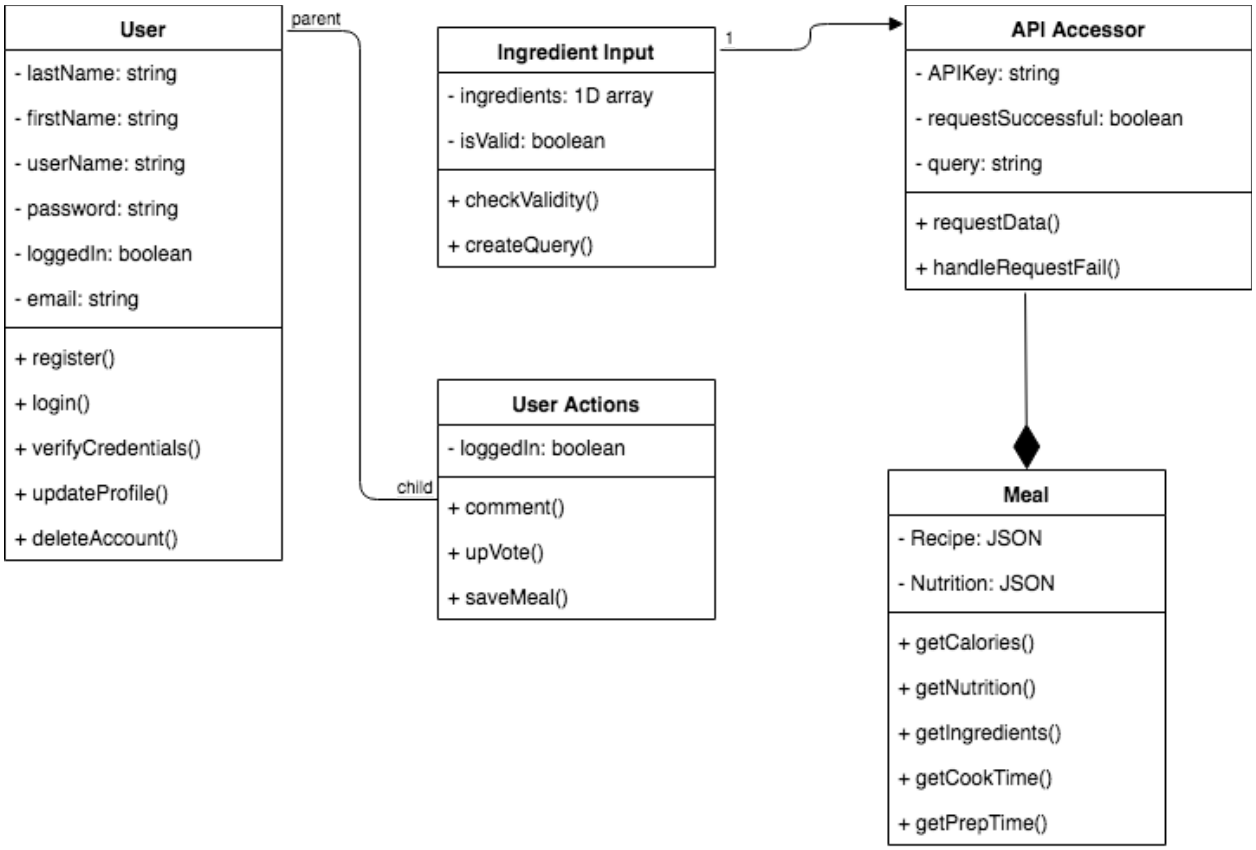
Interaction (System Sequence) Diagrams



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.



Design Patterns

From Wikipedia…"In software architecture, **publish–subscribe** is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize published messages into classes without knowledge of which subscribers, if any, there may be. Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are."

The publish-subscribe pattern applies to our project because Recipe King publishes recipes from the Edamam API to the user, who in this case is a "subscriber". We are not generating the data itself, but we help retrieve and help sort the data that the user specifically requests.

OCL Contract Specification

Context: IngredientInput::checkValidity(ingredients)

Precondition: List of ingredients entered.

Postcondition: Accept the ingredients for the query or notify the user that one or more ingredients needs to be revised.

Context: IngredientInput::createQuery(ingredients)

Precondition: List of ingredients entered.

Precondition: Ingredients are valid.

Postcondition: Create the query from the user-provided ingredients.

Context: UserActions::comment()

Precondition: User is on webpage of recipe on which he/she wishes to comment.

Preconditon: User types out and submits a comment.

Postcondition: Comment is posted to the webpage of the recipe and sent as a record to the database.

Context: UserActions::upVote()

Precondition: User is on recipe page for the recipe he/she wishes to rate.

Postcondition: Rating is saved averaged and displayed next to the meal and the rating received is stored in the database.

Context: UserActions::save()

Precondition: User is on recipe page for the recipe he/she wants to save.

Postcondition: The recipe is saved to the user's profile and to the database.

Context: APIAccessor::requestData()

Precondition: The input list contains valid ingredients and all the ingredients are in at least one meal in the API database.

Postcondition: The query is constructed and a call is made to the API for recipe data.

Context: APIAccessor::handleRequestFail()

Precondition: The request for recipe data fails.

Postcondition: The user is notified of the failure and given a message that indicates why the request failed.

Context: Meal::getCalories()

Precondition: The API call was successful with results.

Postcondition: Calories for the meal are returned.

Context: Meal::getNutrition()

Precondition: The API call was successful with results.

Postcondition: Nutritional information for the meal is returned

Context: Meal::getIngredients()

Precondition: The API call was successful with results.

Postcondition: Ingredients for the meal are returned

Context: Meal::getCookTime()

Precondition: The API call was successful with results.

Postcondition: Cook time for the meal is returned

Context: Meal::getPrepTime()

Precondition: The API call was successful with results.

Postcondition: Prep time for the meal is returned

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

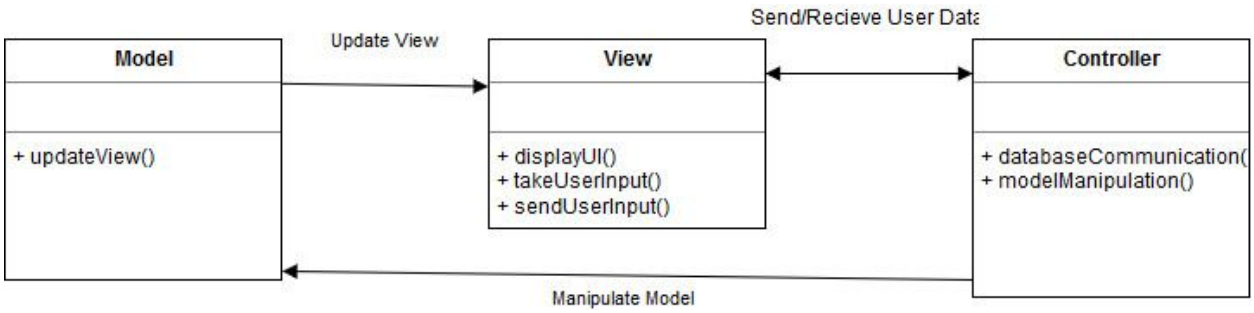
Architectural Style

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 Order:

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

Algorithms and Data Structures

Algorithms

In 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();
```

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

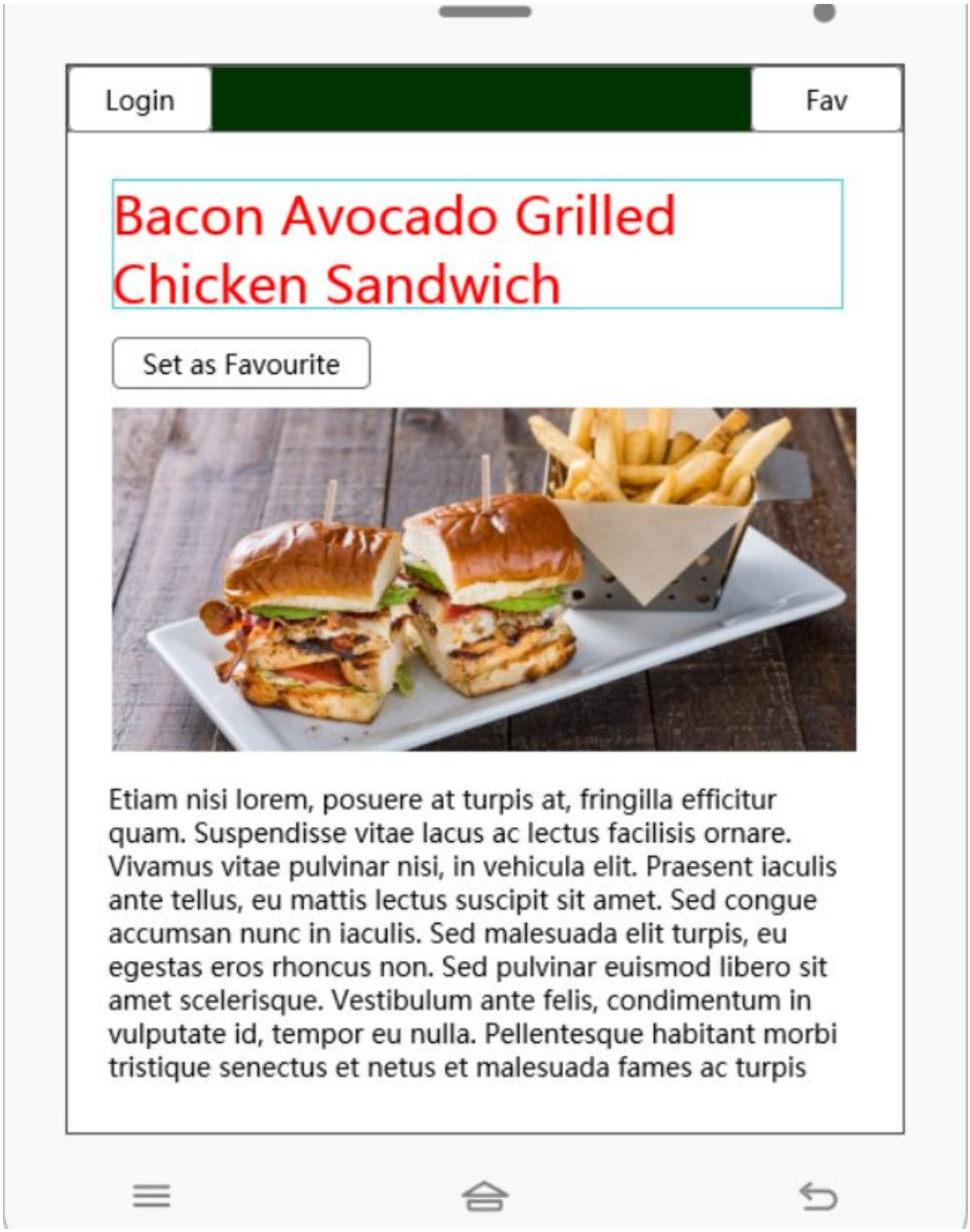
Filter1

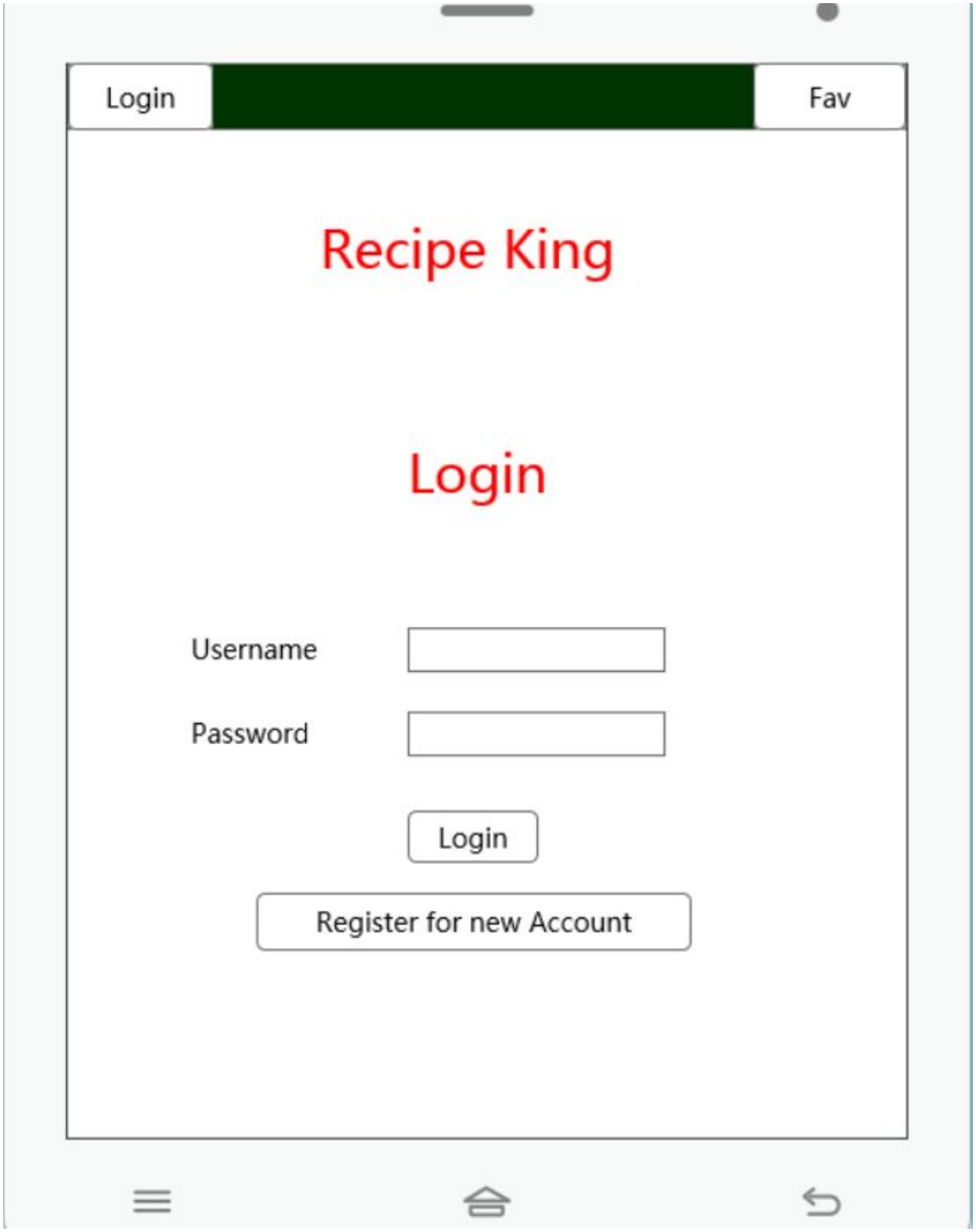
Filter2

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

Login | Fav

Favourite Recipes

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	

Design of Tests

A.Test Case

Test Case TC-1: Ingredients List
Test Designed by: Shayne Smither **Test Executed by**: Jiayu Xia **Module Name**: Recipe Search Screen **Test Title**: Ingredients List Test **Description**: Test ingredients list corresponding to the ingredients input by the user **Preconditions**: The web page has loaded correctly with all UI elements displaying properly **Test Steps**: 1.Input the same ingredient name to the recipe search textfield and click add button Test Data: "Chicken" "Chicken" Expected Result: None of the duplicated ingredient name should be displayed more than once. Actual Result: As Expected Status: Pass 2.Delete all the ingredient names added in the list Expected Result: The list of input ingredients is cleaned up. Actual Result: As Expected Status: Pass 3.Input ingredient names to the recipe search textfield and click add button Test Data: "Chicken" "Rice" "Spice" Expected Result: A list of input ingredients showed below. Actual Result: As Expected Status: Pass **Postconditions**: The system returns a list of recipes corresponding to the ingredients input by the user

Test Case TC-2: Recipe Search
Test Designed by: Shayne Smither **Test Executed by**: Jiayu Xia **Module Name**: Recipe Search Screen **Test Title**:Recipe Search Test **Description**: Generate a list of recipes from a database based off of a provided list of ingredients **Preconditions**: A list of recipes corresponding to the ingredients input is provided **Test Steps**: 1.Get the list of ingredients in the recipe search area and click on Search Recipe Button Test Data: "Chicken" "Rice" "Spice" Expected Result: A list of recipes based off of the provided list of ingredients showed below. Actual Result: As Expected Status: Pass 2.Click on the headers of columns. Expected Result: The list of recipes should be sorted in the chosen column by number and alphabet. Actual Result: As Expected Status: Pass **Postconditions**: The system returns a list of recipes corresponding to the ingredients input by the user

Test Case TC-3: Navigate Recipe
Test Designed by: Shayne Smither **Test Executed by**: Jiayu Xia **Module Name**: Recipe Search Screen, Recipe Info Screen **Test Title**:Navigate Recipe Test **Description**: Navigate to the recipe view and generate the detail of the recipe including content and photo **Preconditions**: A list of recipes corresponding to the ingredients input is provided **Test Steps**: 1.Get the list of recipes and click on a recipe in the table Test Data: "5 spice Broken Rice porridge"

Expected Result: A view of recipe information detail based off of the provided list of ingredients.
Actual Result: As Expected
Status: Pass
2.Click on the Back Button
Expected Result: Go back to the recipe search Screen
Actual Result: As Expected
Status: Pass

Postconditions: A view is generated in detail along with photos of the prepared meal.

Test Case TC-4: Profile Management

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.

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.

History of Work

The goal at the outset of this project was to create a functional web application that would provide more features and give users additional flexibility compared to most solutions around currently. A lot of the web applications catered toward our target user base do not do a very good job of minimizing the work that a user must do in order to find his or herself a suitable meal. A majority of these applications, run on basic keyword searches and often do not allow for valuable filtering or sorting. We have created a simple but powerful web application that will better serve users in their quest for finding a meal that matches the ingredients they have on-hand.

The creation of this web application has progressed at a steady pace. After the basic schematic of this idea was worked out, and a suitable API identified, we began formal diagramming. We came up with functional requirements and use cases, interaction and class diagrams, and added on top of these until we were satisfied with the robustness of our application. The team is not well-aligned in terms of technology stack familiarity, so we had an uphill battle with regard to coding the application. There was difficulty in properly getting the data from the server-side via API call and formatting it correctly on the client-side, but these issues were resolved and the UI and server-side is working as intended.

Current Status

We have created a website with the main functionality that we set out to create. That is, a user can input a list of ingredients and receive back a list of recipes relevant to their list. The application connects to the Edamam API to generate the list of recipes, so we don’t need to store our own recipe data and users do not need to create any accounts to use it. The website looks clean and uncluttered. Recipes can be clicked on to view a complete list of necessary ingredients, a larger picture of the completed dish, the calorie count, and the serving size. The recipes can also be sorted in several different ways, such as alphabetically, number of ingredients, servings, and calories.

Future Work

Software is rarely ever truly finished, and can be improved, tweaked, and polished. Recipe King has the basic functionality that we set out to complete, but more functionality can be added, such as account creation and usage. Specifically, with more time we would like to add our own database to store user accounts, comments made by our users about recipes, and allow users to save recipes for their own future use.

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

https://developer.edamam.com/edamam-docs-recipe-api

6. Wikipedia Definition of Publish-Subscribe Pattern:

https://en.wikipedia.org/wiki/Publish-subscribe_pattern