San Francisco State University

SW Engineering CSC648 / 848 Spring 2020

“STOCK UP”

Team 103

Siddhita Chimote schimote@mail.sfsu.edu

Kevin Dizon-Cua

Calvin Tam

Sara Tama

Corey Russ

Matthew Ibrahim

Cassidy Mcskimming

Jose Pascual

Milestone 2

Revision 2 (04/14/2020)

Revisions

1. 03/24/2020
    - First submission
2. 04/14/2020
    - Function requirement priority adjustment

Data Definitions V2

Data items

1. Storables

   DESCRIPTION: Items that can be stored in the refrigerator.

   a. Ingredients

      DESCRIPTION: Components of the storable in the case of multiple ingredients in one storable.

   b. Quantity

      DESCRIPTION: Amount left in one unit of the storable.

   c. Nutrients

      DESCRIPTION: General term for a storable's vitamins, protein, calories, and other nutritional values.

   d. Purchase Date

      DESCRIPTION: The date the storable was purchased.

   e. Expiration Date

      DESCRIPTION: The date of expiration of the storable.

   f. Owner

      DESCRIPTION: The user who purchased the item.

   g. Intolerable

      DESCRIPTION: Warnings if the storable contains any allergens or intolerable ingredients for users who include it in their profile.

   h. Price

      DESCRIPTION: How much the storable costs.

2. Non-Storables

   DESCRIPTION: Items that cannot be stored in the refrigerator.

3. User

   DESCRIPTION: Person who uses the refrigerator.

   a. Name

      DESCRIPTION: Identifier name of the current user of the refrigerator.

   b. Consumption

      DESCRIPTION: Daily, weekly, monthly, and total nutrient intake of the user.

   c. Saved Recipes

      DESCRIPTION: List of the user's saved recipes.

   d. Recipe History

      DESCRIPTION: List of the recipes the user has tried before.

   e. Intolerances

      DESCRIPTION: List of user's allergy and intolerance information.

   f. Meal Plan

      DESCRIPTION: select daily plan for a week of meals based on storables listed inside fridge

Data structures

1. Recipes

   DESCRIPTION: A list of storables which are required to prepare a particular meal.

   a. Ingredients

      DESCRIPTION: Components of the recipe.

   b. Nutrients

      DESCRIPTION: General term for the recipe's vitamins, protein, calories, and other nutritional values.

   c. Cooking Time

      DESCRIPTION: The amount of time it would take to cook the recipe.

   d. Instructions

      DESCRIPTION: List of steps on how the user will cook the recipe.

    e. Price

       DESCRIPTION: Total cost of the recipe.

    f. Servings

       DESCRIPTION: Amount of servings the recipe makes.

2. Refrigerator

   DESCRIPTION: Object that contains all the storable objects.

3. Shopping List

   DESCRIPTION: Object that contains all objects that the user needs to purchase.

    a. Price

       DESCRIPTION: Cost of each object, and cost of total objects that the user needs to purchase.

4. Expired Storables List

   DESCRIPTION: Object that contains all objects that need to be removed from the refrigerator.

5. Low Stock List

   DESCRIPTION: Object that contains all the low stock objects.

6. Household

   DESCRIPTION: Group of users who use the refrigerator.

Functional Requirements V2

Scales

Priority level

1. Must have
2. Desired
3. Opportunistic

Function 1: User input

- Users input information about adding an item to the refrigerator or removing an item from the freezer.
  - As for clarity, the project considers the receipt (invoices) from “Safeway”.
- *PRIORITY:* 1
- *USER STORY:* Persona 1-7

Function 1.1: Reading receipt

- Input should be very convenient. As a main use case, it will be the receipt input out of the grocery store using OCR API (Safeway receipt).
- The app will sort the items into storables and Non-storables, adding only the storables to the user’s fridge inventory.
- *PRIORITY:* 2

Function 1.2: Manually adding / removing an Item

- It should provide a way to Add/Remove certain items (“Remove 2 apples”) when items are consumed or discarded or purchased with no receipt.
- *PRIORITY:* 1

Function 1.3: Barcode input

- Barcode input (2D and QR codes). As the fall-back case when our choice of major input does not work, it should allow the manual input by website menu selection (e.g. “Add”).
- PRIORITY: 2

Function 1.4: User Sign Up/Info

- Users will be able to sign up for a master fridge(multiple users can belong to one fridge). When signing up, they can input any dietary restrictions, or allergies they want to be alerted about or avoid in search results for recipes
- *PRIORITY:* 1
- *USE CASE:*
  - Users will consistently be putting things and taking things out of their refrigerator. This requires both main methods and alternative methods of inputting all of the data in order for the app to monitor everything appropriately.
  - Every single user will operate / interact with user input of the application.

Function 2: Inventory

- Users can keep an inventory (one fridge per user) of what they have.
  - It is highly recommended to access external product databases to look for product nutrition information for the given input item. For now, we are able to provide product nutrition information via 3rd party APIs. We can get the specific name of the item from the receipt, and then look up the item using the 3rd party APis provided in our document.
- *PRIORITY:* 1
- *USER STORY:* Persona 1-7

Function 2.1: Look for product nutrition information

- The app will access external databases via 3rd party APIs to provide nutrition information for given input items purchased by the user.
- *PRIORITY:* 2

Function 2.2: Find recipes based on inventory

- Using a 3rd party database of recipes, use the current items to determine what recipes the user is able to make.
- If there are missing ingredients from the recipe, we can notify and make it clear to the user what ingredients are missing and suggest that they add the missing items to a shopping list.
- *PRIORITY:* 2
- *USE CASE:*
    - A working professional has a busy life. Sometimes they are not good at keeping track of certain things; they need some way to keep track of information such as what ingredients / foods they have at home.
    - Our app will allow the user to keep track of what kind of goods they have in their fridge so that it is possible to know what they have even with busy lives.
    - A professional cook may want to try a variety of recipes; he / she wants to branch out and try all kinds of recipes to expand his / her palette. To get inspiration, our app will allow him / her to search for recipes based on current itesm they have in the fridge / at hand.

Function 3: Expiration date log

- Users to be notified before items expire or low on stock level.
- *PRIORITY:* 1
- *USER STORY:* Persona 4-6

Function 3.1: Expiration input

- May have to manually input the expiration date of the product when we add to our database.
- *PRIORITY:* 2

Function 3.2: Expiring soon notification

- Stock up will notify users when an item is close to its expiration date.
- *PRIORITY:* 1

Function 3.3: Running low notification

- Stock up will notify users when an item is close to running out.
- *PRIORITY:* 2
- *USE CASE:*
  - A parent will often create meals for their kids from the fridge. Ideally, parents would like to feed their kids food that isn't expired and are of healthy quality.
  - Therefore, a useful feature to have would be to keep an expiration date recorded on the app.

Function 4: Shopping list

- Users can make / keep a shopping list to keep a record of goods they need to pick up next time they go to the store.
- *PRIORITY:* 2
- *USER STORY:* Persona 1-7

Function 4.1: Shopping list history

- Users will be able to see past shopping lists.
- *PRIORITY:* 2

Function 4.2: Create shopping list based on recipe

- User inputs a recipe and the ingredients needed to make the dish, then a shopping list is generated upon entering all ingredients needed.
- *PRIORITY:* 3
- *USE CASE:*
  - Users will consistently be consuming the storables that are coming into their refrigerator, eventually they need to replenish what they've taken out.
  - Many people make shopping lists, all users would likely use / benefit from a feature like this.
  - Users sometimes end up at the store and with no list, they forget what exactly it is that they need to pick up. Stock up solves this problem for its users. There will be no excuse to not have a shopping list.

Function 5: Recipes

- Users are able to save recipes, search for recipes, etc.
- *PRIORITY:* 2
- *USER STORY:* Persona 1-7

Function 5.1: Missing from recipe

- Provides the user with access to quickly see what ingredients they need in order to use the selected recipe.
- *PRIORITY:* 3
- *USE CASE:*
  - A person may really love a certain recipe. For easy access, maybe they'll want to save the recipe. Our app will allow the user to save their favorite recipes; each user can save their own favorite list of recipes.

    - This favorite recipe list will show the recipes the user saved, and the user will be able to click on any of the recipes to see what ingredients are needed for the specific recipe.

Function 6: User report

- Users to view the report (per user) regularly on their consumption of items
- *PRIORITY:* 2
- *USER STORY:* Persona 5

Function 6.1: View user info

- Simulating a profile type view where you can review a user's allergy information, consumption summary, saved recipes, etc.
- *PRIORITY:* 2

Function 6.2: View user consumption

- e.g. how much milk do they consume per month, etc
    - Highly recommended to report their nutrition consumption by nutrition category (calories, fat, protein, vitamin, …). It is advisable to access external product databases to look for product nutrition information.
    - Users really care about the accuracy of the report.
- *PRIORITY:* 2
- *USE CASE:*
    - A Nutritionist will closely monitor their food and nutrition consumption and will likely hold the accuracy of the data provided to a high standard.
    - Parents might also want to view reports of what the entire family might be consuming weekly.

Function 7: Meal plan

- Users can create a meal plan for themselves.
- *PRIORITY:* 2
- *USER STORY:* Persona 1-7

Function 7.1: Create a meal plan

- Stock up will search through saved recipes first to include in meal plans since users often have routine eating habits.
- *PRIORITY:* 2
- *USE CASE:*
    - Some user's might be trying to watch their diet, and desire to make meal plans that follow along with their dietary restrictions.
    - For user's that are too busy, meal plans might help for saving time throughout the day.

Handling technical challenges

All of our functional requirements are technologically feasible within our deadline. For example, we can read the input of our receipt by reading the barcode via a 3rd party API, or using Google Cloud's Vision AI (https://cloud.google.com/vision).

We can also calculate the nutrient information of certain items simply by using another 3rd party API, such as spoonacular(https://spoonacular.com/food-api). We will categorize our items as storables/non-storables. If we have something like 'MLK' on the receipt to represent milk but milk is represented as 'milk', when no results are shown when attempting to fetch 'MLK' from the database, we can suggest to the user to type in the item that they might have meant, and then our app will show suggestions/results.

High Level UI Mockups and Storyboards

UI mockups

https://builderx.io/app/53eydeeqlssgggwwg0og0c0swwgwws

Storyboard 1

- *PERSONA:* 1
- *USER ACTIONS:*
    - check inventory for low-stock / expired storables
    - check consumption info for all users
    - create shopping list based on recipe and allergies / intolerances
    - add storables
    - consume / discard storables
    - find recipe based on storables in stock / cooking time / servings / nutrition
- *VIEWS:*
    - view inventory of fridge (including low stock and expired storables)
    - view consumption info per user
    - view consumption info of all users
    - add storable to inventory
    - search recipe (with advanced filter)
    - view recipe ingredients (showing which storables are in stock)
    - view shopping list
    - add storable to shopping list

Storyboard 2

- *PERSONA:* 3
- *USER ACTIONS:*
    - check inventory for low-stock / expired storables
    - check consumption info for all users
    - add storables
    - consume / discard storables
- *VIEWS:*
    - view inventory of fridge (including low stock and expired storables)
    - view inventory per user
    - view consumption info per user
    - view consumption info of all users
    - add storable to inventory

Storyboard 3

- *PERSONA:* 5
- *USER ACTIONS:*
    - check inventory for low-stock / expired storables
    - check consumption info for all users
    - create shopping list based on recipe and allergies / intolerances
    - add storables
    - consume / discard storables
    - find recipe based on cooking time / servings / nutrition

- *VIEWS:*
    - view inventory of fridge (including low stock and expired storables)
    - view consumption info per user
    - view consumption info of all users
    - add storable to inventory
    - search recipe (with advanced filter)
    - view recipe ingredients (showing which storables are in stock)
    - view shopping list
    - add storable to shopping list

Storyboard 4

- *PERSONA:* 6
- *USER ACTIONS:*
    - check inventory for low-stock / expired storables
    - check consumption info for all users
    - create shopping list based on recipe
    - add storables
    - consume / discard storables
    - find recipe based on storables in stock / cooking time / servings
- *VIEWS:*
    - view inventory of fridge (including low stock and expired storables)
    - view consumption info per user
    - view consumption info of all users

    - add storable to inventory
    - search recipe (with advanced filter)
    - view recipe ingredients (showing which storables are in stock)
    - view shopping list
    - add storable to shopping list

High Level Architecture, Database Organization

Database organization

fridges

Column name	Data type	Attributes
fridge_id	UNSIGNED INT	PRIMARY KEY, AUTO INCREMENT

users

Column name	Data type	Attributes
user_id	UNSIGNED INT	PRIMARY KEY, AUTO INCREMENT
fridge_id	UNSIGNED INT	FORIEGN KEY REFERENCES fridges(fridge_id), INDEX, NOT NULL
name	VARCHAR(64)	NOT NULL, UNIQUE INDEX
role	VARCHAR(64)	DEFAULT NULL

storables

Column name	Data type	Attributes
storable_id	UNSIGNED INT	PRIMARY KEY, AUTO INCREMENT
nutrition_id	UNSIGNED INT	FOREIGN KEY REFERENCES nutrition(nutrition_id), DEFAULT NULL
name	VARCHAR(64)	UNIQUE INDEX, NOT NULL
image	TEXT	DEFAULT NULL

recipes

Column name	Data type	Attributes
recipe_id	UNSIGNED INT	PRIMARY KEY, AUTO INCREMENT
nutrition_id	UNSIGNED INT	FOREIGN KEY REFERENCES nutrition(nutrition_id), NOT NULL
name	VARCHAR(64)	-UNIQUE INDEX, NOT NULL
image	TEXT	DEFAULT NULL
servings	UNSIGNED TINYINT	NOT NULL
cooking_time	UNSIGNED SMALLINT	NOT NULL, DEFAULT 0 (in minutes)

instructions	TEXT	DEFAULT NULL

nutrition

Column name	Data type	Attributes
nutrition_id	UNSIGNED INT	PRIMARY KEY, AUTO INCREMENT
intolerances	TEXT	DEFAULT NULL (JSON)
calories	UNSIGNED SMALLINT	NOT NULL
carbohydrates	UNSIGNED SMALLINT	NOT NULL (in grams)
fat	UNSIGNED SMALLINT	NOT NULL (in grams)
protein	UNSIGNED SMALLINT	NOT NULL (in grams)

recipe_ingredients

Column name	Data type	Attributes
storable_id	UNSIGNED INT	FOREIGN KEY REFERENCES storables(storable_id), INDEX, NOT NULL
recipe_id	UNSIGNED INT	FOREIGN KEY REFERENCES storables(recipe_id), INDEX, NOT NULL
quantity	REAL	NOT NULL, DEFAULT 1

inventory

Column name	Data type	Attributes
inventory_id	UNSIGNED INT	PRIMARY KEY, AUTO INCREMENT
fridge_id	UNSIGNED INT	FOREIGN KEY REFERENCES fridges(fridge_id), NOT NULL, INDEX
storable_id	UNSIGNED INT	FOREIGN KEY REFERENCES storables(storable_id), NOT NULL
purchase_date	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP
purchased_price	UNSIGNED SMALLINT	NOT NULL (in pennies)
state		NOT NULL, INDEX, DEFAULT 'stored'
remaining	REAL	NOT NULL, DEFAULT 1 (percent)

expiration

Column name	Data type	Attributes
inventory_id	UNSIGNED INT	FOREIGN KEY REFERENCES inventory(inventory_id), NOT NULL
expiration_date	TIMESTAMP	INDEX, DEFAULT NULL

consumption

Column name	Data type	Attributes
user_id	UNSIGNED INT	FOREIGN KEY REFERENCES users(user_id), INDEX, NOT NULL
inventory_id	UNSIGNED INT	FOREIGN KEY REFERENCES inventory(inventory_id), INDEX, NOT NULL
consumed_date	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP

recipe_history

Column name	Data type	Attributes
user_id	UNSIGNED INT	FOREIGN KEY REFERENCES users(user_id), INDEX, NOT NULL
recipe_id	UNSIGNED INT	FOREIGN KEY REFERENCES recipe(recipe_id), NOT NULL
consumed_date	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP

recipe_favorites

Column name	Data type	Attributes
user_id	UNSIGNED INT	FOREIGN KEY REFERENCES users(user_id), INDEX, NOT NULL
recipe_id	UNSIGNED INT	FOREIGN KEY REFERENCES recipe(recipe_id), NOT NULL
favorited_date	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP

Search / filter architecture and implementation

1. Searching can be done on recipes/storables (in inventory or otherwise) via SQL's LIKE clause.

2. Filtering based on specified intolerances (i.e. lactose intolerance = no recipes / storables that contain dairy will be shown) via SQL’s NOT LIKE clause.

APIs

Endpoint	Method	Parameters	Returns
/users	GET	name: string	user_id: uint32
/users	POST	name: string, fridge_id: uint32, role: string?	user_id: uint32
/fridges	GET	user_id: uint32	fridge_id: uint32
/fridges	POST		fridge_id: uint32
/inventory	GET	fridge_id: uint32, begin: uint32?, limit: uint32?. state: string?	inventory_ids: uint32[]
/inventory	POST	fridge_id: uint32, storable_id: uint32, price: uint16, quantity: uint16?, expiration: int32?	inventory_id: uint32
/inventory	PATCH	inventory_id: uint32, remaining: float	
/inventory/consume	PATCH	user_id: uint32, inventory_id: uint32	

Technical challenges

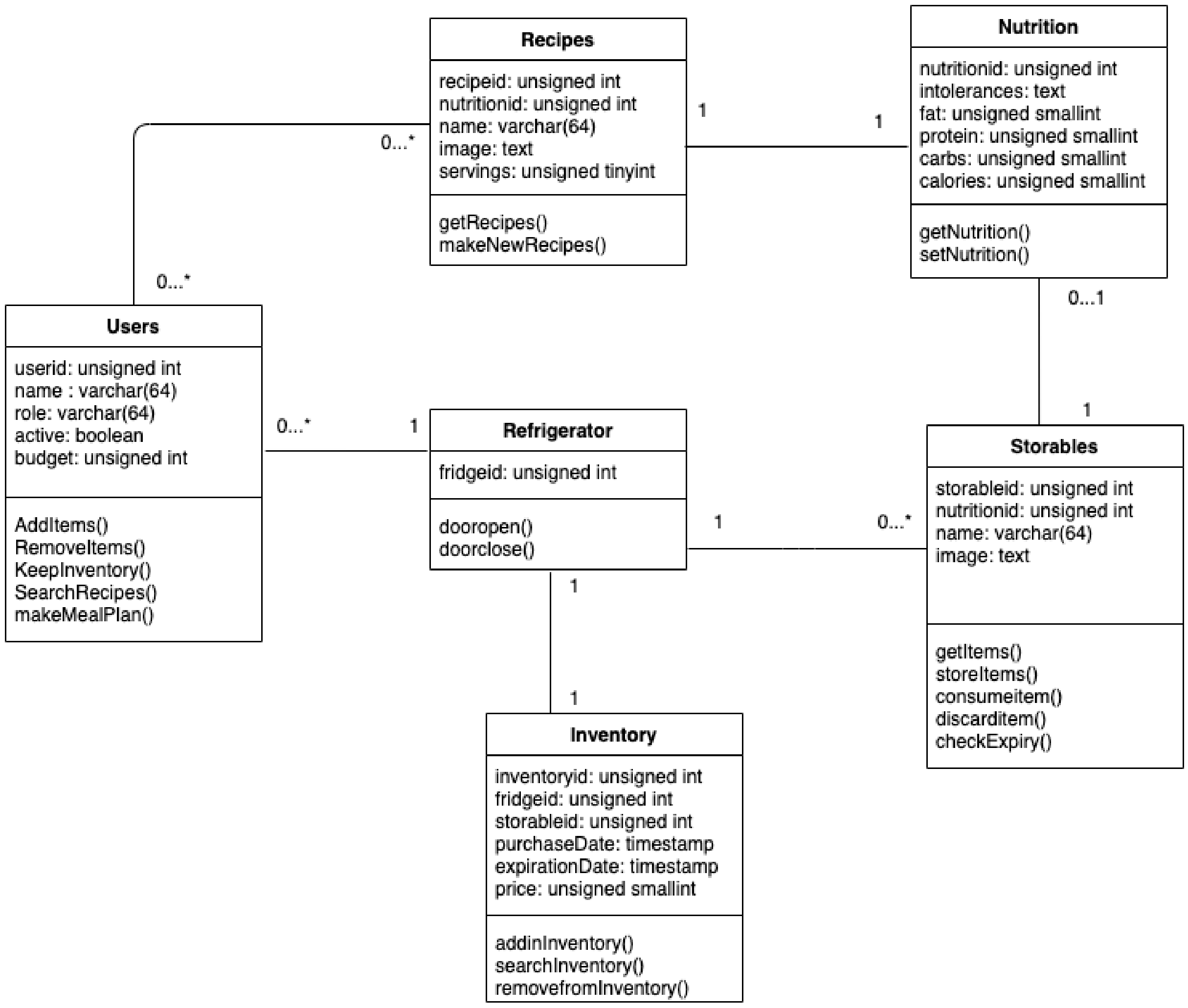
1. Keep the database small enough but also fast enough to be used by at least a handful of people.
2. Determine how to effectively populate the storables/recipes table via edamam.
3. Efficiently calculate nutritional statistics (another column in the users table or compute it every time?)
4. Handle duplicate storables with varying expiration dates.
5. Determine how to handle search queries (support for wildcards? sorting?)
6. Non-consumables must be filtered out, but consumables not yet in the db must not be. This means a call to the edamam API for every item not in our database. Keeping

track of non-storables seen in another table would likely reduce the amount of edamam API calls significantly.

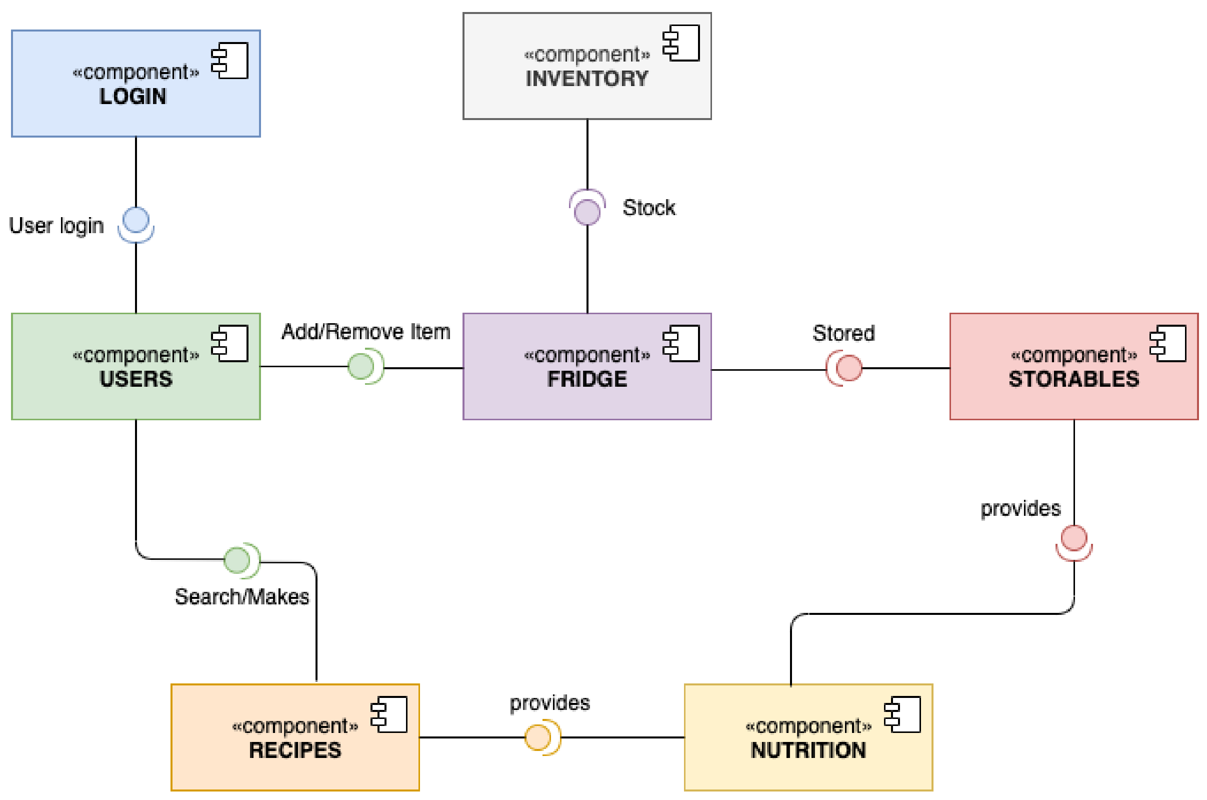
7. Nutrient information is provided by edamam and stored per item/recipe in the database, but for user-generated recipes will it be calculated based on ingredients or via input by the user?

High Level UML Diagrams

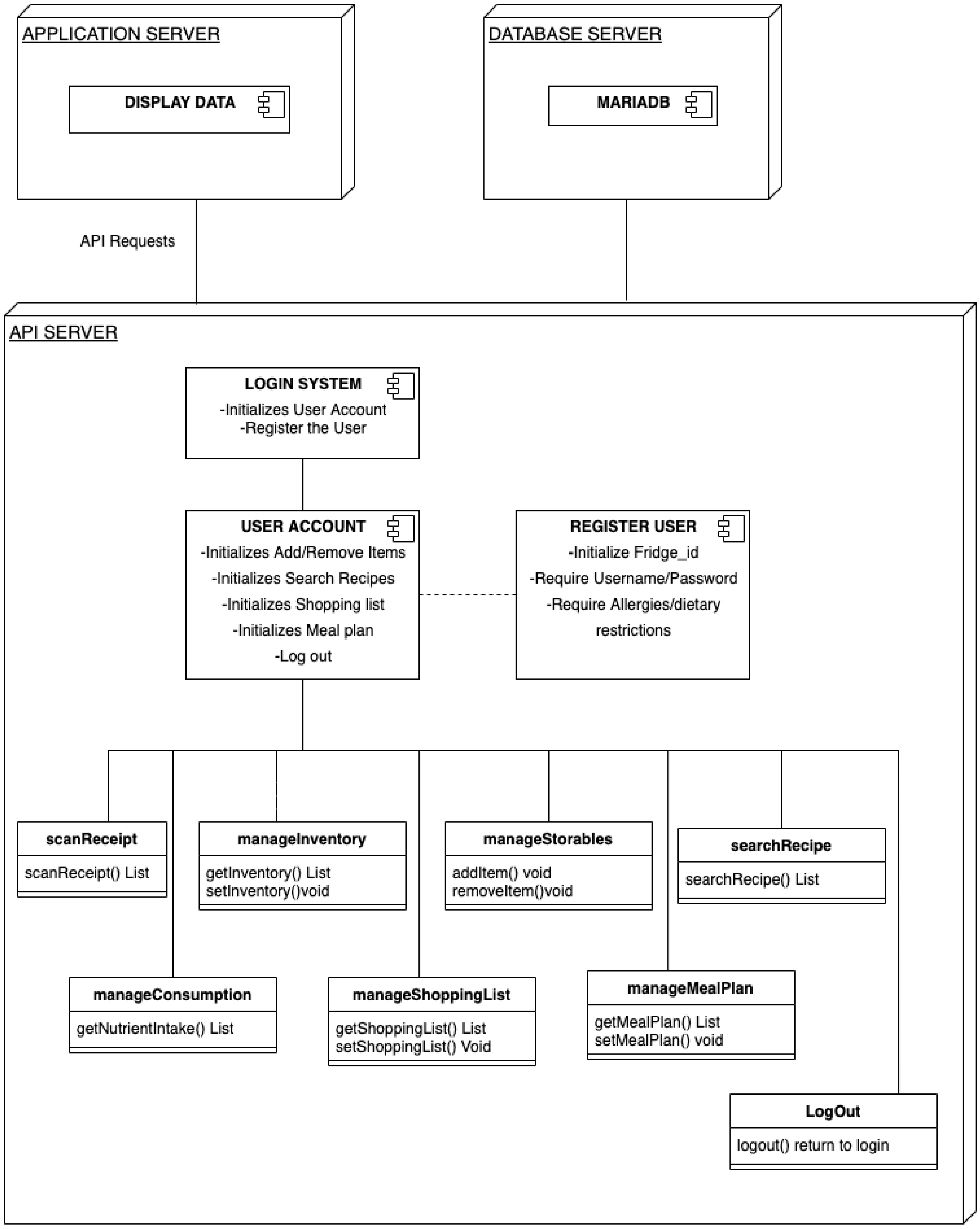
UML class diagram



UML component diagram



UML deployment diagram



Risk Assessments

Risk 1: Github usage

- *TYPE:* Teamwork risk
- *RESOLUTION:*
    - To resolve this issue, we will make the git branching as easy as possible to understand. This will enable all of our team members to understand how to branch properly and how to merge properly.

Risk 2: Learning React

- *TYPE:* Skills risk
- *RESOLUTION:*
    - To resolve this issue, we will work together to learn React by completing tutorials.
    - Additionally, if there is a member who knows React already, they will share their knowledge with the rest of the team and ensure we all can understand.

Risk 3: Schedules among team members

- *TYPE:* Scheduling risk
- *RESOLUTION:*
    - In order to solve this issue, the scrum master/team leader will ensure that each team member has an assigned task, and follow up with that member to ensure that the task will be completed on time.

Risk 4: Properly obtained legal rights

- *TYPE:* Legal / content risk
- *RESOLUTION:*

    - First, we will not sell this product.
    - Second, we will ensure that each product we add will have a license which allows us to use it.

Risk 5: Inaccurate assessment of requirement difficulty

- *TYPE:* Skills/technical risk
- *RESOLUTION:*
    - In order to avoid this risk, we will need to know exactly how we will implement a requirement before we add it to the project.
    - This will ensure we can fulfill the requirement without being overwhelmed with too many new tasks that accompany a requirement.

Risk 6: Lack in clarity of requirements

- *TYPE:* Teamwork risk
- *RESOLUTION:*
    - Some of the requirements are vague, thus allowing each team member to interpret them differently.
    - In order to solve this, we will describe exactly what we will do with each of the requirements so every team member will be on the same page.

Project management

Our team’s project management is definitely developing as tasks increasingly become more complex and dependent on others. For Milestone 2, our team gathered to review the M2 document together, and divided up tasks based on the roles of each of our team members.

Our focus was drawn towards making sure enough help was assigned to the tasks that require more attention. As our team has been adjusting to all communication being conducted virtually, we have largely relied on discord and zoom for group discussion. Regular group meetings have been a priority to keep track of progress and ensure everything is on track towards meeting our self assigned deadlines. We’ve used a variety of tools to help us complete all these requirements, working with draw.io and Visual paradigm to draw UML diagrams, github to share source code and keep a track of project’s status, google docs to share documentation, figma for UI mockups, etc.

Moving forward, we will continue to develop our project management choices as tasks keep changing along with our environments. So far our communication has been good, we hope to keep that going until the project is officially complete.