SW Engineering CSC648/848 Summer 2024

ScholarEats

Team 4:

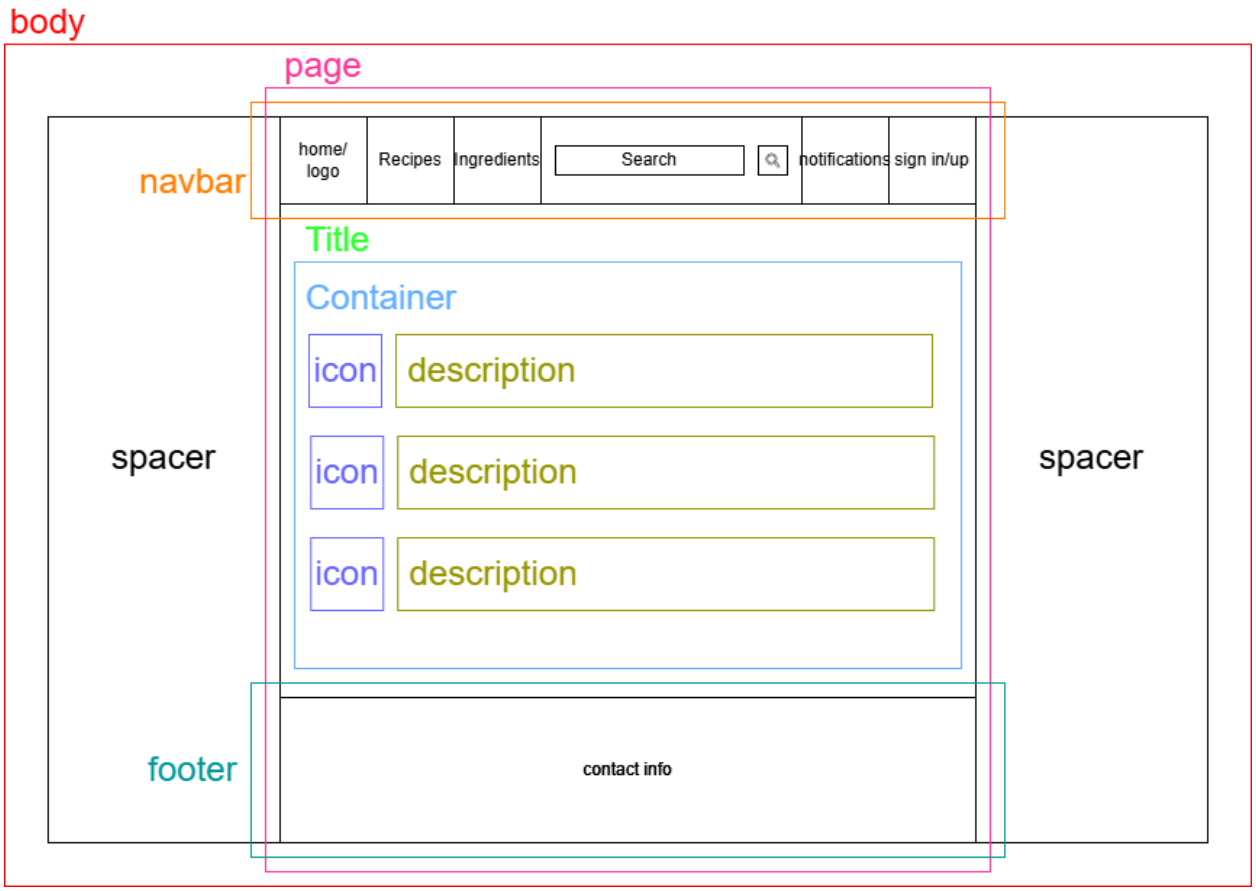
Angelo Arriaga (Team Lead - aarriaga1@sfsu.edu)
Donovan Taylor (Front-End Lead)
Hancun Guo (Front-End)
Tina Chou (Front-End)
Edward McDonald (Back-End Lead)
Karl Carsola (Back-End)
Sai Bavisetti (Database)
Maeve Fitzpatrick (Docs Editor)
Sabrina Diaz-Erazo (GitHub Master)

Milestone 2

07/09/24

History (Revisions)	
2024-07-09	Milestone 2 Submission

1. Data Definitions



Database

activity_log
log_id
user_id
action
timestamp

admin
admin_id
uuid
username
password_hash
verification_status
accessibility

email_log
email_log_id
user_id
email
timestamp

expired_products
ingredient_id
Name
expiration_date
quantity

has
user_id
university_id

ingredients
ingredient_id
name

manages
admin_id
user_id

notifications
notification_id
user_id
message
is_read

recipe_ingredient
recipe_id
ingredient_id

recipes
Unnamed: 0
recipe_name
prep_time
cook_time
total_time
servings
yield
ingredients
directions
rating
cuisine_path
nutrition
timing
img_src
dietary restrictions

	calories
	protein
	fat
	fiber
	difficulty
	cooking tip
	user_id
	MyUnknownColumn

roles
role_id
role_name

sessions
session_id
user_id
session_start
session_end
ip_address
user_agent

store
store_id
Name
ingredient_id
user_id
quantity
expiration_date

university
university_id
name
email_suffix

user_info
user_info_id
user_id
profile_photo
bio
bio

	pronouns
	allergies
	dietary_restrictions
	favorited_recipes
	favorited_ingredients
	university

user_recipes	user_recipe_id
	user_id
	recipe_name
	prep_time
	cook_time
	total_time
	yield
	directions
	img_source
	nutrition
	servings
	dietary_restrictions
	calories
	fat
	protein
	fiber
	difficulty
	cooking_tip
	accessibility

	is_approved
	admin_id

Users	user_id
	uuid
	email
	username
	password_hash
	verification_status
	accessibility
	blacklist
	role_id
	university

2. Initial List of Functional Requirements

PRIORITY 1 - Critical

- Administrators
    1. ADMINISTRATORS SHALL BE ABLE TO AUTHENTICATE THE USERS.
    2. ADMINISTRATORS SHALL BE ABLE TO BLACKLIST/ WHITELIST A USER
    3. ADMINISTRATORS SHALL BE ABLE TO EDIT THE INGREDIENT LIST
    4. ADMINISTRATORS SHALL BE ABLE TO REMOVE USER ACCOUNTS
    5. ADMINISTRATORS SHALL BE ABLE TO FOLLOW THE EXPIRATION DATE CLOSELY AND TAKE THE NECESSARY ACTIONS.
- System
    1. SYSTEM SHALL AUTOMATICALLY ENROLL USERS IN CORRESPONDING UNIVERSITY FOOD PROGRAMS
    2. SYSTEM SHALL BE ABLE TO GENERATE RECOMMENDED RECIPES
    3. SYSTEM SHALL TELL ADMINISTRATORS WHEN AN INGREDIENT HAS RUN OUT OF STOCK
    4. SYSTEM SHALL TELL ADMINISTRATORS WHEN FOOD HAS SPOILED
- Users
    1. USERS SHALL BE ABLE TO CHANGE THEIR PASSWORD
    2. USERS SHALL BE ABLE TO EDIT THEIR USERNAME
    3. USERS SHALL BE ABLE TO FILTER RECIPES BY COOKING AIDS REQUIRED
    4. USERS SHALL BE ABLE TO FILTER RECIPES BY DIETARY RESTRICTIONS
    5. USERS SHALL BE ABLE TO FILTER RECIPES BY DIFFICULTY
    6. USERS SHALL BE ABLE TO FILTER RECIPES BY INGREDIENTS
    7. USERS SHALL BE ABLE TO MAKE ACCOUNTS WITH VALID UNIVERSITY EMAIL.
    8. USERS SHALL BE ABLE TO SET ALLERGIES
    9. USERS SHALL BE ABLE TO SET DIETARY RESTRICTIONS
    10. USERS SHALL BE ABLE TO SORT RECIPES BY CALORIES
    11. USERS SHALL BE ABLE TO SORT RECIPES BY FAT
    12. USERS SHALL BE ABLE TO SORT RECIPES BY FIBER
    13. USERS SHALL BE ABLE TO SORT RECIPES BY PROTEIN
    14. USERS SHALL BE ABLE TO VIEW AVAILABLE INGREDIENTS
    15. USERS SHALL BE ABLE TO VIEW RECIPES

PRIORITY 2 - Desired

- Administrators
    1. ADMINISTRATORS SHALL BE ABLE TO REMOVE RECIPES OR REVIEWS
- Systems
    1. SYSTEM SHALL ENSURE RECIPES ARE AVAILABLE FOR ALL (MAJOR) DIETARY RESTRICTIONS
- Users
    1. USERS SHALL BE ABLE TO FAVORITE INGREDIENTS
    2. USERS SHALL BE ABLE TO FAVORITE RECIPES
    3. USERS SHALL BE ABLE TO REVIEW/RATE RECIPES
    4. USERS SHALL BE ABLE TO SORT RECIPES BY RATING
    5. USERS SHALL BE ABLE TO VIEW OWN FAVORITE RECIPES

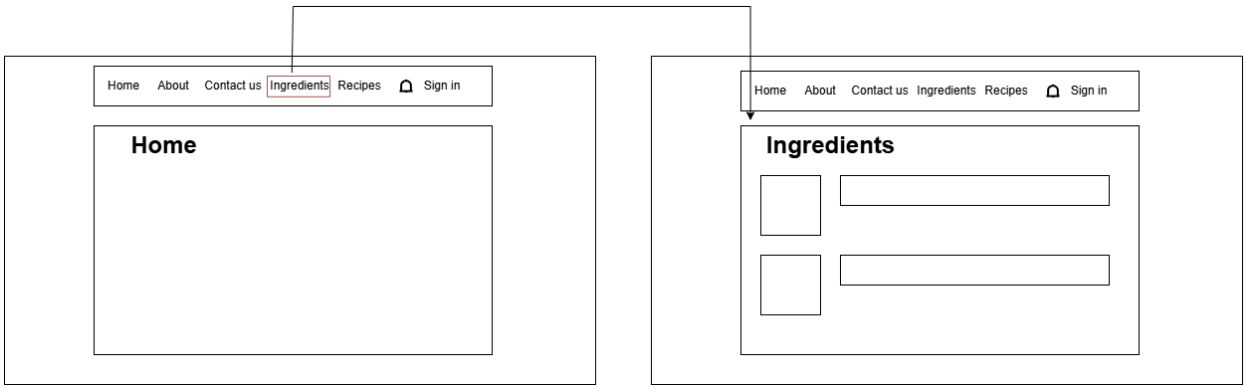
PRIORITY 3 - Opportunistic

- Administrators
    1. ADMINISTRATORS SHALL BE ABLE TO DELETE USER POSTED RECIPES
    2. ADMINISTRATORS SHALL BE ABLE TO MAKE UNIVERSITY-WIDE ANNOUNCEMENTS
    3. ADMINISTRATORS SHALL BE ABLE TO PROMOTE RECIPES
    4. ADMINISTRATORS SHALL BE ABLE TO VIEW ALL USER-REPORTS
- Users
    1. USERS SHALL BE ABLE TO ADD THEIR PREFERRED PRONOUNS
    2. USERS SHALL BE ABLE TO ALLOW NOTIFICATIONS
    3. USERS SHALL BE ABLE TO BLOCK OTHER USERS
    4. USERS SHALL BE ABLE TO CHANGE PROFILE PHOTO
    5. USERS SHALL BE ABLE TO EDIT PROFILE BIO
    6. USERS SHALL BE ABLE TO REPORT PHOTOS OR REVIEWS
    7. USERS SHALL BE ABLE TO REPORT USER ACCOUNTS
    8. USERS SHALL BE ABLE TO SHARE RECIPES
    9. USERS SHALL BE ABLE TO SUBMIT RECIPES
    10. USERS SHALL BE ABLE TO SWITCH BETWEEN LIGHT AND DARK MODE
    11. USERS SHALL BE ABLE TO TRANSFER ENROLLMENT IN UNIVERSITY FOOD PROGRAMS
    12. USERS SHALL BE ABLE TO UPLOAD PHOTOS OF THE RECIPES THEY COOK FOR REVIEWS
    13. USERS SHALL BE ABLE TO VIEW HISTORY OF RECEIVED RECIPES
    14. USERS SHALL BE ABLE TO VIEW OTHER USER’S FAVORITE INGREDIENTS
    15. USERS SHALL BE ABLE TO VIEW OTHER USER’S FAVORITE RECIPES

3. UI Mockups and Storyboards

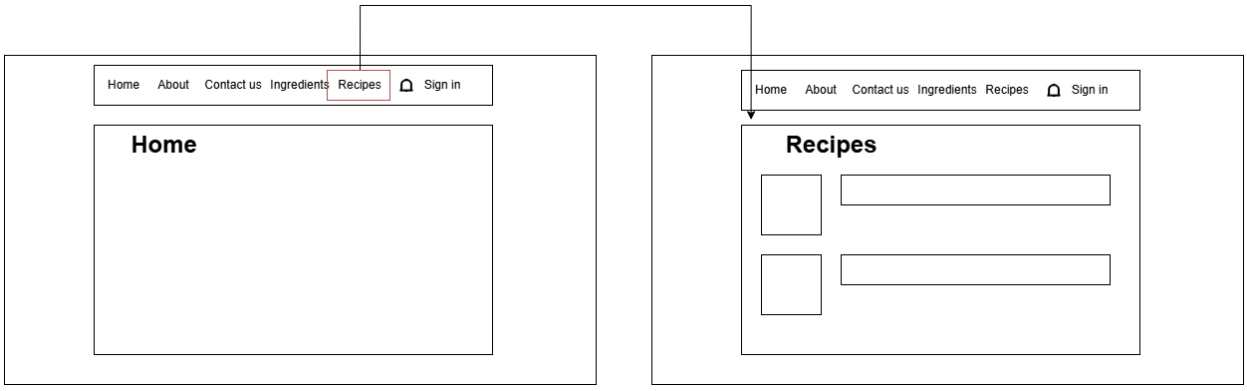
Use Case 1: Meal Planning:

Since the student is already using his university's grocery program, all he needs to do is go online and look at the recipes that are provided for his school's inventory. Once he signs up, he can browse a myriad of recipes using ingredients supplied by Gator Groceries.



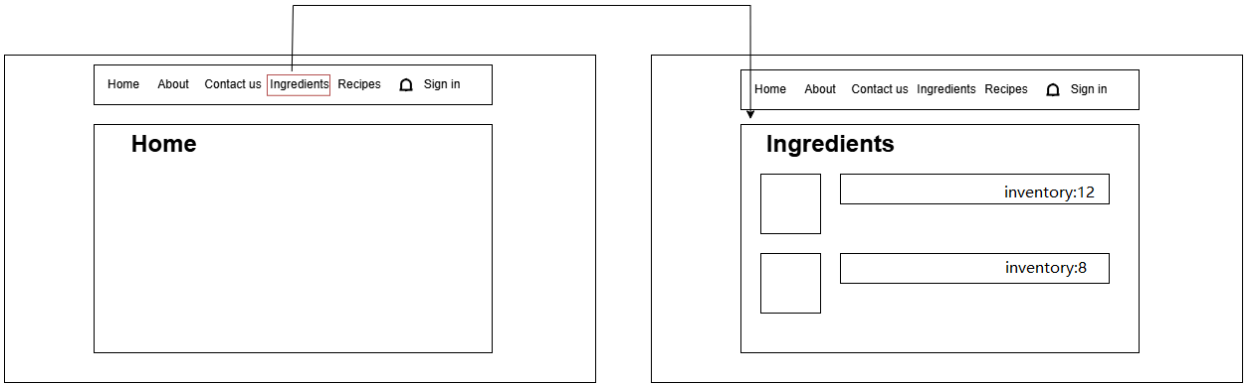
Use Case 2: Money Saving:

Jane utilizes an online service (ScholarEats), which shows her the food available this week at her school's free food program. Along with this information, she also receives a few auto-generated recipes. This further encourages her to go to her university's free food program.



Use Case 3: Waste Reduction:

Julie then finds out about ScholarEats and accesses it on her phone. Julie is amazed with the variety of recipes that are available and how every ingredient from Gator Groceries is utilized in some way. Now, when she goes to Gator Groceries, she can get exactly the type and amount of ingredients she needs without having to worry about any of them going to waste. She also likes that there is an inventory feature so she does not have to worry about showing up to Gator Groceries and having to find out that everything is gone.



Use Case 4: Health Consciousness:

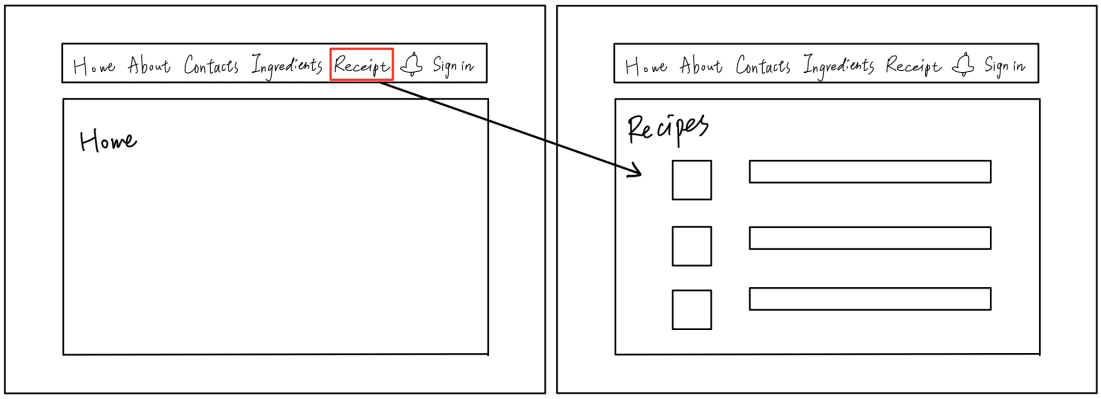
Jack logged in and checked available ingredients. He selected a low-carb recipe. The platform provided necessary ingredients and step-by-step cooking instructions. Jack picked up the ingredients from campus grocery hub and cooked his meal. This program helped him maintain a healthy diet.

Use Case 5: Student Engagement:

Joe looks into his automatically-enrolled account, and opts in for the next week's groceries. After opting in himself, Joe posts about the program on social media, which catches their friend Dwight's attention; he is intrigued by the program, looks into it himself and also opts in for the groceries.

Use Case 6: Time Saving:

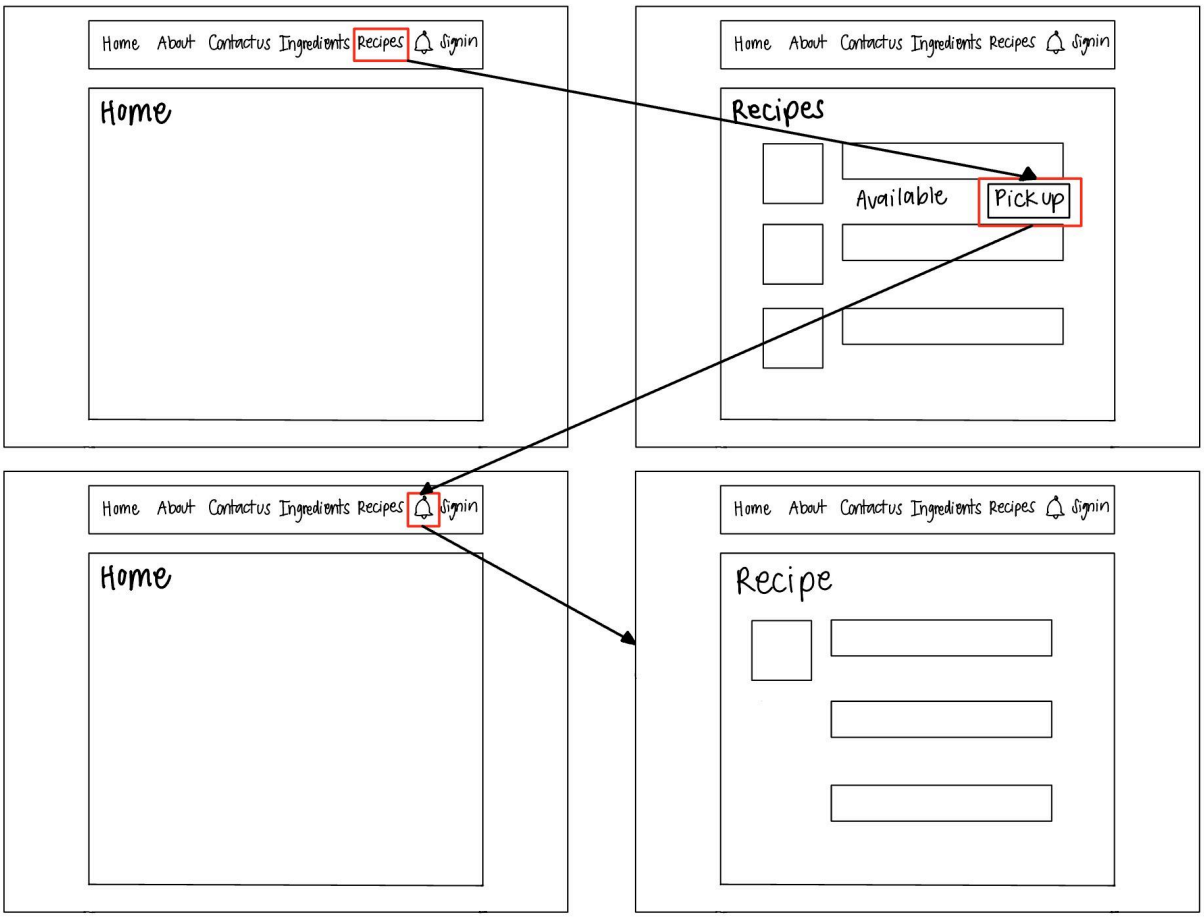
Jessie, a student who is currently attending full-time at her university and a part-time worker at a convenience store, has a hard time balancing her school and work life. Despite the many efforts that she has made to manage her time properly she always falls short with the time she has for herself. Luckily, she found that she can save more time with the use of ScholarEats where Jessie can easily get recipes based on the groceries that her university has. With this, she is able to be stress free and able to spend less time on finding what to make.



Use Case 7: Cooking Education:

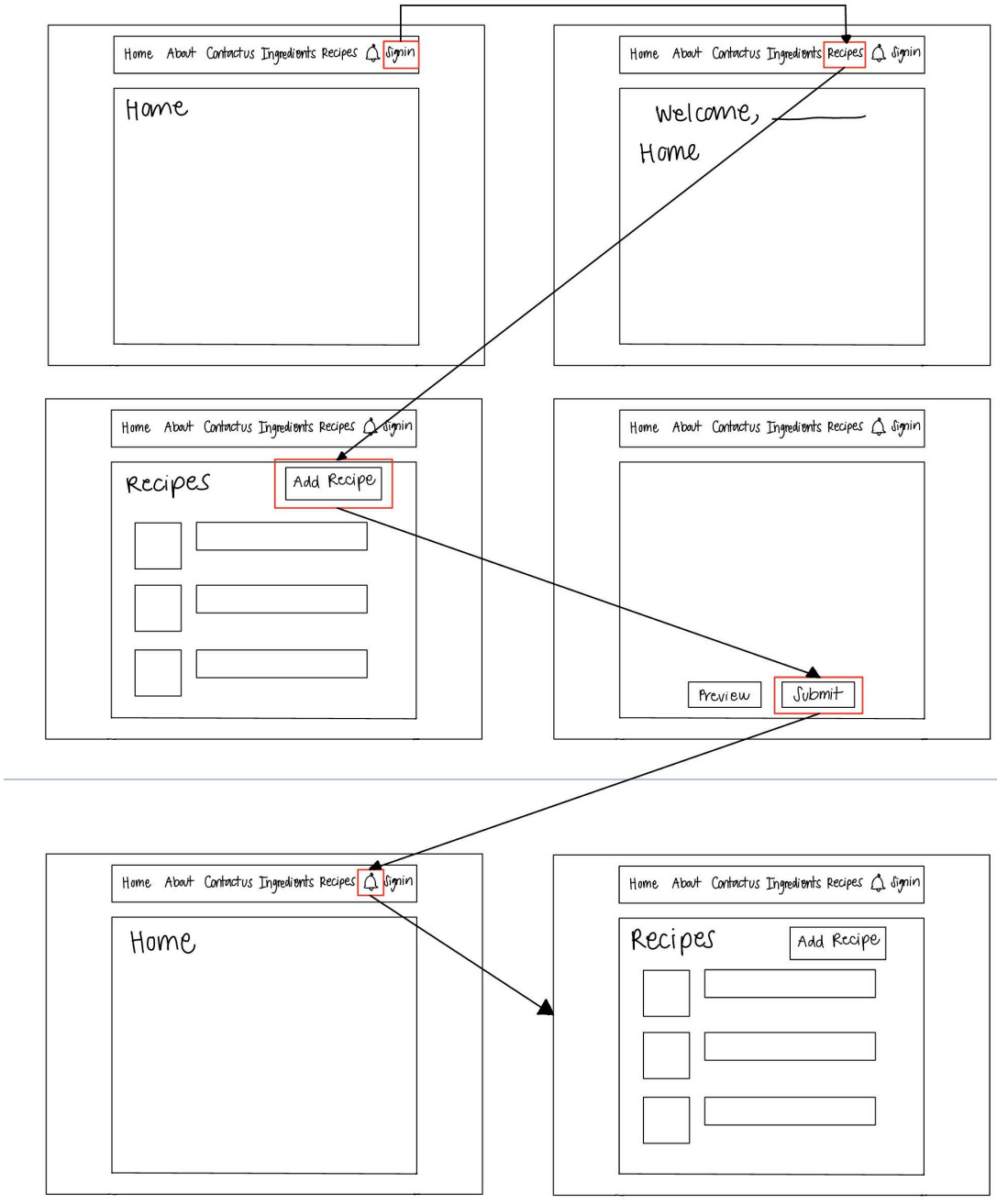
Olivia can use this app that her school manages locally to see which recipes have ingredients readily available within the university's food pantry. Olivia confirms her intention of picking up a set of

ingredients earlier in the day and gets notified when it is ready to pick up. Olivia picks up the ingredients. Olivia goes to her dorm and the app gives her step by step instructions on how to cook.



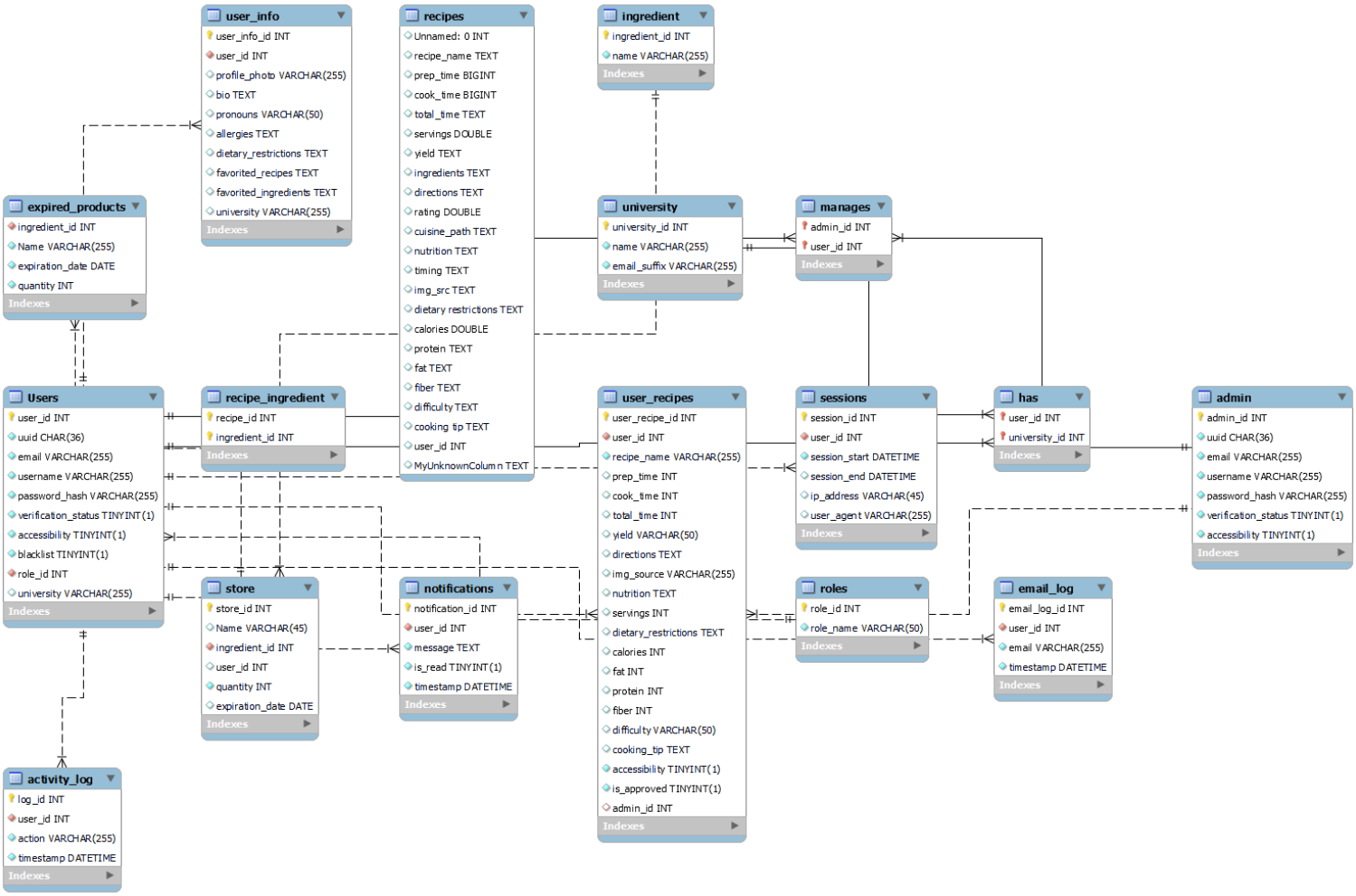
Use Case 8: Community Recipe Submission:

Motivated by a desire to engage with her peers and enrich the campus dining experience, she logs into the ScholarEats platform, eager to introduce others to her culinary creation. The platform's user-friendly interface for the recipe submission process includes boxes for the ingredient list, step-by-step instructions, and a place to submit optional photos of the dish. It is well organized and easy to navigate. She appreciates the option to preview her entry before finalizing it. Upon submission, the ScholarEats system quickly processes the recipe, displaying a confirmation message that reassures Emily that her contribution has been received and is under review. Emily receives a notification when her recipe is live, adding a sense of accomplishment and fostering a deeper sense of community involvement.

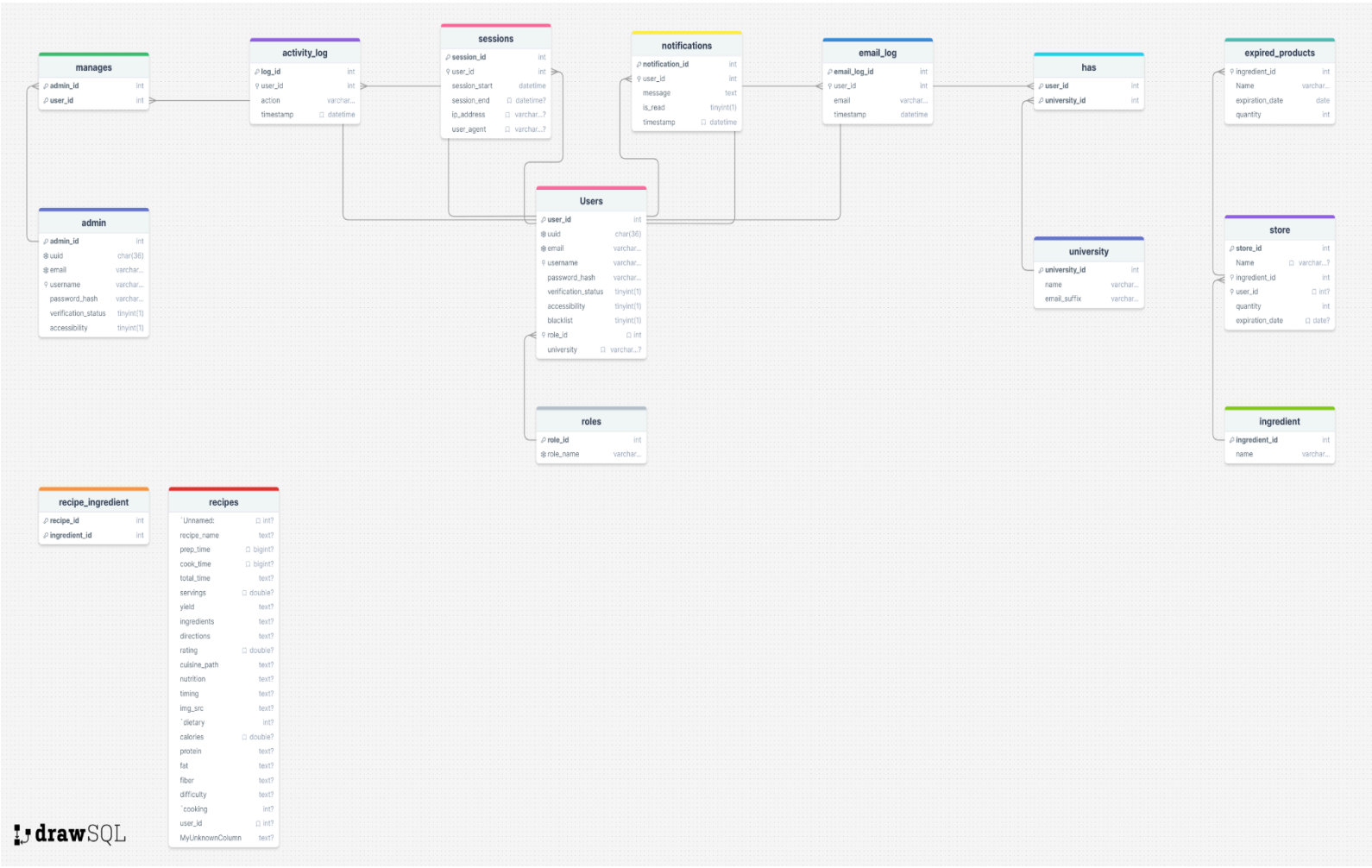

Home About Contactus Ingredients Recipes Signin
Home
Home About Contactus Ingredients Recipes Signin
Welcome,
Home
Home About Contactus Ingredients Recipes Signin
Recipes
Add Recipe
Home About Contactus Ingredients Recipes Signin
Preview
Submit
Home About Contactus Ingredients Recipes Signin
Home
Home About Contactus Ingredients Recipes Signin
Recipes
Add Recipe

4. High level database architecture and organization

EER Diagram :



ERD Diagram :


manages
admin_id int
user_id int
activity_log
log_id int
user_id int
action varchar...
timestamp datetime
sessions
session_id int
user_id int
session_start datetime
session_end datetime?
ip_address varchar...?
user_agent varchar...?
notifications
notification_id int
user_id int
message text
is_read tinyint(1)
timestamp datetime
email_log
email_log_id int
user_id int
email varchar...
timestamp datetime
has
user_id int
university_id int
expired_products
ingredient_id int
Name varchar...
expiration_date date
quantity int
admin
admin_id int
uuid char(36)
email varchar...
username varchar...
password_hash varchar...
verification_status tinyint(1)
accessibility tinyint(1)
Users
user_id int
uuid char(36)
email varchar...
username varchar...
password_hash varchar...
verification_status tinyint(1)
accessibility tinyint(1)
blacklist tinyint(1)
role_id int
university varchar...?
university
university_id int
name varchar...
email_suffix varchar...
store
store_id int
Name varchar...?
ingredient_id int
user_id int?
quantity int
expiration_date date?
roles
role_id int
role_name varchar...
ingredient
ingredient_id int
name varchar...
recipe_ingredient
recipe_id int
ingredient_id int
recipes
`Unnamed: int?
recipe_name text?
prep_time bigint?
cook_time bigint?
total_time text?
servings double?
yield text?
ingredients text?
directions text?
rating double?
cuisine_path text?
nutrition text?
timing text?
img_src text?
`dietary int?
calories double?
protein text?
fat text?
fiber text?
difficulty text?
`cooking int?
user_id int?
MyUnknownColumn text?
drawSQL

5. High Level APIs and Main Algorithms

Sorting APIs:

- Endpoint: router.get('/sortByCaloriesAsc')
    - Returns recipes sorted in ascending order based on their calorie content.
- Endpoint: router.get('/sortByCaloriesDesc')
    - Returns recipes sorted in descending order based on their calorie content.
- Endpoint: router.get('/sortByProteinAsc')
    - Returns recipes sorted in ascending order based on their protein content.
- Endpoint: router.get('/sortByProteinDesc')
    - Returns recipes sorted in descending order based on their protein content.
- Endpoint: router.get('/sortByFatAsc')
    - Returns recipes sorted in ascending order based on their fat content.
- Endpoint: router.get('/sortByFatDesc')
    - Returns recipes sorted in descending order based on their fat content.
- Endpoint: router.get('/sortByFiberAsc')
    - Returns recipes sorted in ascending order based on their fiber content.
- Endpoint: router.get('/sortByFiberDesc')
    - Returns recipes sorted in descending order based on their fiber content.

Filtering APIs:

- Endpoint: router.get('/filterByDiet/:restriction')
    - Filters recipes based on a specific dietary restriction indicated by :restriction.
    - Returns recipes that match the specified dietary restriction.
- Endpoint: router.get('/filterByCookingAids/:aid')
    - Filters recipes based on specific cooking aids required indicated by :aid.
    - Returns recipes that require the specified cooking aid.
- Endpoint: router.get('/filterByDifficulty/:level')
    - Filters recipes based on difficulty level indicated by :level.
    - Returns recipes that match the specified difficulty level.

Fetch Available Ingredients API

- router.get('/')
    - Retrieves a list of available ingredients from the database.
    - It queries the store table to fetch ingredient_id, quantity, and name of each ingredient joined with the ingredient table.
    - It then renders an ingredients page with the retrieved data, displaying each ingredient with an associated image, name, and quantity.

Check Expired Items API

- Endpoint: router.get('/checkExpired')

    - Updates the expired status of food items in the store table based on their expiration_date.
    - It sets expired = TRUE for items where expiration_date is less than or equal to the current date (YYYY-MM-DD format).

Remove Out of Stock Items API

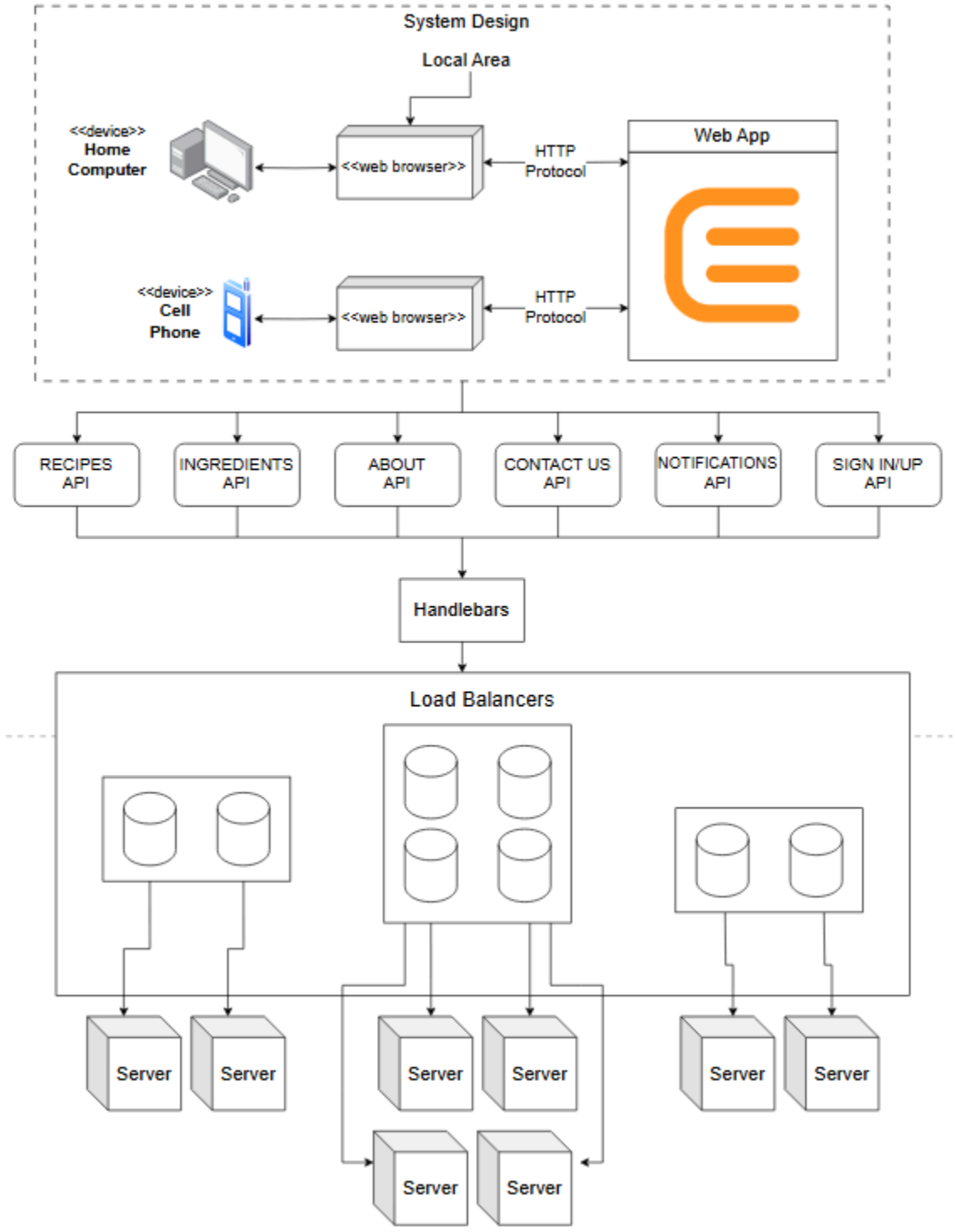
- Endpoint: router.get('/checkOutOfStock')
    - Deletes items from the store table where quantity_available is less than or equal to zero, indicating that the item is out of stock.

User APIs

- Endpoint: router.post(‘/register’)
    - Registers a new user their email and password and stores their credentials into the database.
- Endpoint: router.post(‘/login’)
    - Allows a user who created an account to log in with their email and password.
- Endpoint: router.post(‘/logout’)
    - Logs out a user that is currently logged in by terminating their session token.
- Endpoint: router.post(‘/change-password’)
    - Allows the user to change their password by giving the current password and then giving the new password.
- Endpoint: router.post(‘/change-username’)
    - Allows the user to change their username.
- Endpoint: router.post(‘/set-allergies)
    - Allows the user to set their allergies in their profile.
- Endpoint: router.post(‘/set-dietary-restrictions)
    - Allows the user to set their dietary restrictions in their profile.

6. System Design

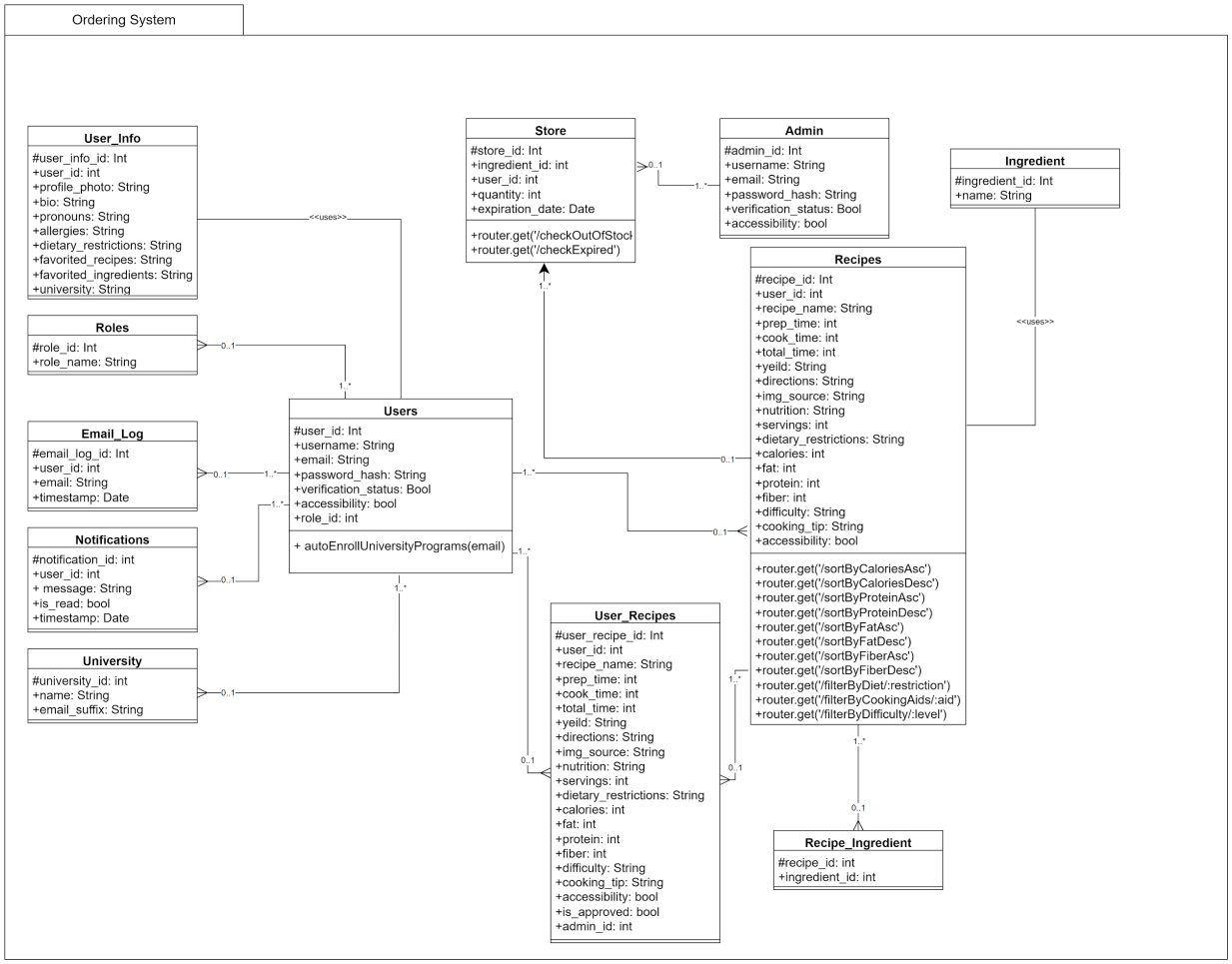
Scalability Illustration Diagram:



Summary of System Architecture Design:

Our system architecture involves every user utilizing either a computer or mobile device that has access to a web browser. That web browser accesses our web app, ScholarEats, over HTTP protocol. The web app interacts with several APIs, including the following tabs: Recipes, Ingredients, About, Contact Us, Notifications and Sign In/Up. The API requests are processed by Handlebars, which then sends the requests to the different load balancers. The load balancers distribute to whatever server is available.

UML Diagrams:



Summary of UML Diagram:

The main entities are ‘Users’, ‘User_Info’, ‘Email_Log’, ‘University’, ‘Store’, ‘Admin’, ‘Recipes’, and ‘Ingredient’.

The ‘User_Info’ class stores detailed user information, such as ‘profile photo’, ‘bio’, ‘pronouns’, ‘allergies’, ‘dietary restrictions’, ‘favorited ingredients’, ‘favorited recipes’, and the ‘university’ the user belongs to. It has a one-to-one association with the ‘Users’ class.

The ‘Users’ class contains basic user information, including ‘username’, ‘email’, ‘password hash’, ‘verification status’, and ‘role ID’. It also has a method to automatically enroll users in university programs based on their email domain. The decision to make the User and User_Info entities separate classes was made in order to simplify maintenance.

The Roles class assigns a role to each user, which is linked through the ‘role_id’. Having Roles as its own class will help with scalability if future roles are ever created.

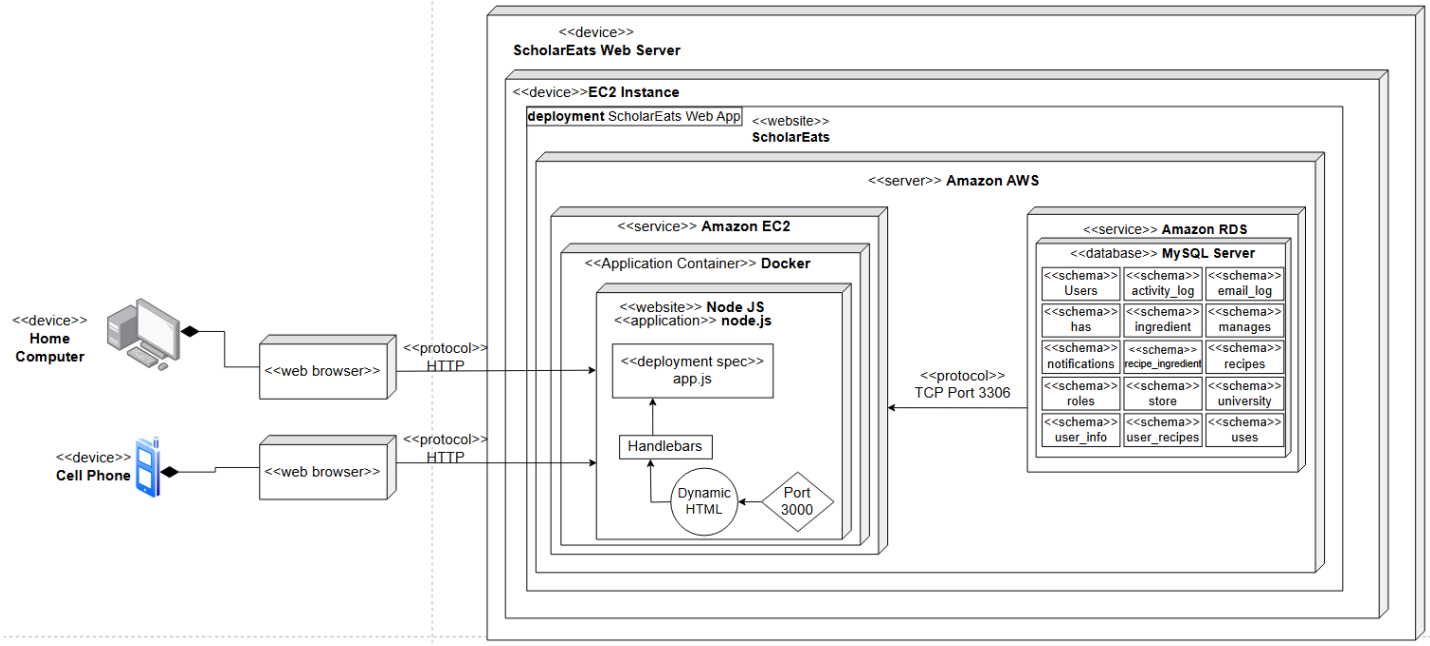
The ‘University’ class contains details about different universities, including their ‘names’ and ‘email suffixes’, which are used for auto-enrollment. The ‘Admin’ class stores admin details such as ‘username’, ‘email’, ‘password hash’, ‘verification status’, and ‘accessibility’. By keeping these entities in separate classes, this design helps to maintain data integrity. Also, making the Admin entity separate from a regular User entity was chosen for security reasons.

The ‘Store’ class manages inventory, including ‘ingredient IDs’, ‘quantities’, and ‘expiration dates’. It contains methods to check for out-of-stock items and expired ingredients and has a one-to-many relationship with the ‘Users’ class. This class is intended to help maintain the inventory in a centralized and consistent way.

The ‘Recipes’ class captures recipe details like ‘recipe name’, ‘preparation time’, ‘cook time’, ‘total time’, ‘yield’, ‘directions’, ‘image source’, ‘nutrition’, ‘servings’, ‘dietary restrictions’, ‘calories’, ‘fat’, ‘protein’, ‘fiber’, ‘difficulty’, ‘cooking tips’, and ‘accessibility’. It includes several methods for sorting and filtering recipes based on these attributes. The ‘User_Recipes’ class links users to their recipes, capturing details such as ‘servings’, ‘directions’, ‘ingredients’, and ‘admin approval status’. Having these various filters and sort options helps to optimize data retrieval.

The ‘Ingredient’ class lists possible ingredients, while the ‘Recipe_Ingredient’ class maps ingredients to recipes, forming a many-to-many relationship between ‘Recipes’ and ‘Ingredient’. By separating these classes, we can include information, like preparation instructions or quantity for a particular ingredient in a particular recipe without affecting the definition/function of the Recipe or Ingredient as a whole.

7. High Level Application Network and Deployment Design



8. Identify actual key risks for your project at this time

- Skills Risks
    - Inexperience making website applications
        - For our website application it will be the first time that many of us will be building a website application from scratch. Previous experiences often had us building on top of a pre-existing project that had some critical dependencies pre-configured.
        - To remedy this, individuals who had in-depth knowledge of specific portions of the project will take brief sections of meetings in order to inform and train other members of the team in critical components, such as the usage of Handlebars and the functionality we could incorporate when it comes to designing our page rendering.
- Schedule Risks
    - Fast-Tempo of Summer
        - This class is a compressed class. It takes place over about two months as opposed to the usual three. This means individual milestones will be much closer together and thus features will have to be quickly implemented.
        - Moving forward tasks will be distributed and supervised through the use of Notion. By listing all requirements ahead of time and assigning them out with individuals in charge of them we can achieve the end product in future milestones.
    - Team Leader very heavy schedule
        - The team lead is taking a total of 15 units over the summer. This severely limits his ability to be present to all tasks being completed.
        - The team lead will have a lighter schedule following the completion of Milestone 2 as 6 of the 15 units will have concluded. This paired with the delegation of tasks through Notion and sitting with team leads to discuss requirements will lead to smoother workflows.
- Technical Risks:
    - Testing procedures have vague checklist requirements
        - Our designed workflow incorporates a final step of testing procedures before a Pull Request (PR) in GitHub could be created. Due to the nature of the project being in early development, these testing checklists are vague and subject to change.
        - To remedy this we introduced a new Git branch called “TestingGuidance” which will be utilized to pass out updated checklists to the master branch as the prototype becomes more developed. This will enable everyone to be able to pull the most recent guidance directly within their branches to minimize confusion.
- Teamwork Risks:

    - Code Review must be tested before-hand individually and then reviewed by 1 of 2 people
        - Our designed workflow's finalization process starts with the maintainer going through the current guidance and ensuring that nothing is immediately broken by their changes. The maintainer then makes a Pull Request (PR) which is approved by either the Team-Lead of the GitHub Master. This process is simultaneously with several branches.
        - To ensure that features are developed and incorporated into the master branch Notion will be used to track the status of each feature. Once the maintainer has finished the branch and believes it to be ready the Notion task will be updated and the Discord team chat will be notified. This will allow all maintainers to be able to see where teammates are at and when they should pull from main to account for changes.
- Legal/Content Risks:
    - Copyrighted content: We have to ensure we have the legal right to use any images on the site
        - This application will be used potentially commercially or as an enterprise product. This means that we have to ensure that all images, resources, and materials used in the creation or operation of ScholarEats either belong to us or we have the legal ability to utilize them. Failure to do so opens up the opportunity for our organization to face legal consequences.
        - We are preemptively avoiding this issue by instituting a policy where we make our own materials or only introduce outside materials with the copyright information alongside it. For an example Institution we are avoiding the usage of the SFSU banner in the footer and will be incorporating a made-up university as an example.

9. Project management

Scholar Eats' development will be managed through the workflow-tracking site "Notion". Tasks will be identified by the Team-Lead and after being defined will be added to the Tasks Section of the Notion Teamspace. At this time the maintainer responsible for that task will be defined and that maintainer will be in charge of implementing the said task. Notion tasks will have statuses updated by the maintainer with where progress stops so that if another maintainer is assigned, they know where the task currently stands. The maintainer will fully and completely implement the task and once the task is believed to have been completed they will examine the checklist proved in the testing guidance. Once that checklist has been examined and verified, they are able to submit a GitHub Pull Request (PR). Once the PR has been made, either the GitHub Master or the Team-Lead will review the PR and either approve or close it. Once approved the PR can be merged into the master branch. At no points will the master branch ever be committed to or altered in any way outside of the procedure above. Once the PR has been successfully merged the "task" is considered complete and will be updated accordingly on Notion.

10. Detailed list of contributions (this section must be done by the team lead)

- Donovan: (Score 9.5/10)
    - Led session to progress Section 3 of Milestone 2
    - Contributed to UI Mockups
    - Incorporated the notification message notifying the user of unimplemented features in the UI
    - Created the Layout to utilize Handlebars for dynamic HTML generation
    - Led a training session to inform others of the functionality and usage of Handlebars
    - Oversaw the creation of all UI pages
- Hancun: (Score 7/10)
    - Contributed to UI Mockups
    - Created the Login Page
    - Added the Forgot Password functionality
    - Added Input Validation
    - Added Remember Me Option
- Tina: (Score 7/10)
    - Contributed to UI Mockups
    - Created the .hbs for the Ingredients Page
    - Created the .hbs for the Recipes Page
    - Created the .hbs for the User Account Page
    - Furthered Account Management
- Edward: (Score 10/10)
    - Led session to tackle Section 4 of Milestone 2
    - Contributed to UI Mockups
    - Created some helper queries to access data from the database
    - Utilized the database queries and APIs to properly pass and render the pages through the defined .hbs files
    - Worked with the defined search parameters to properly build the SQL Query to populate the database.
- Karl: (Score 7/10)
    - Contributed to the High-Level Database Design
    - Implemented the creation of users
    - Implemented the ability to sign in as a user
    - Implemented the ability to change a user’s username
    - Implemented the ability to change a user’s passwords
    - Implemented the ability to set a user’s dietary restrictions
    - Implemented the ability to set a user’s allergies
- Sai: (Score 7/10)
    - Contributed to the High-Level Database Design

    - Wrote a script that handled a majority of the auto-complete requirements for the Minimum Viable Prototype
    - Created some helper queries to access data from the database
    - Helped configure the UI to contain the autocomplete functionality.
    - Assisted with Documentation by completing EER and ERD Diagrams
- Maeve: (Score 9.5/10)
    - Led session to tackle Section 6 of Milestone 2
    - Contributed to UI Mockups
    - Finalized Section 6
    - Finalized Section 7
    - Assisted with the development of frontend.
    - Helped ensure that every team member submitted their submission for Milestone 2
- Sabrina: (Score 9/10)
    - Tested and Reviewed over a dozen individual Pull Requests on GitHub.
    - Contributed to UI Mockups
    - Contributed to the documentation involving the System Design
    - Created and completed the requirements for the page footer of the Minimum Viable Prototype
    - Maintained and updated all relevant credentials