Group 013-06 - Frugal Finder
Project Report

Nathan Keyt
Niki Hellmers
Maxwell Prue
Aieshah Safi
Bennett Fragomeni

Title: Frugal Finder

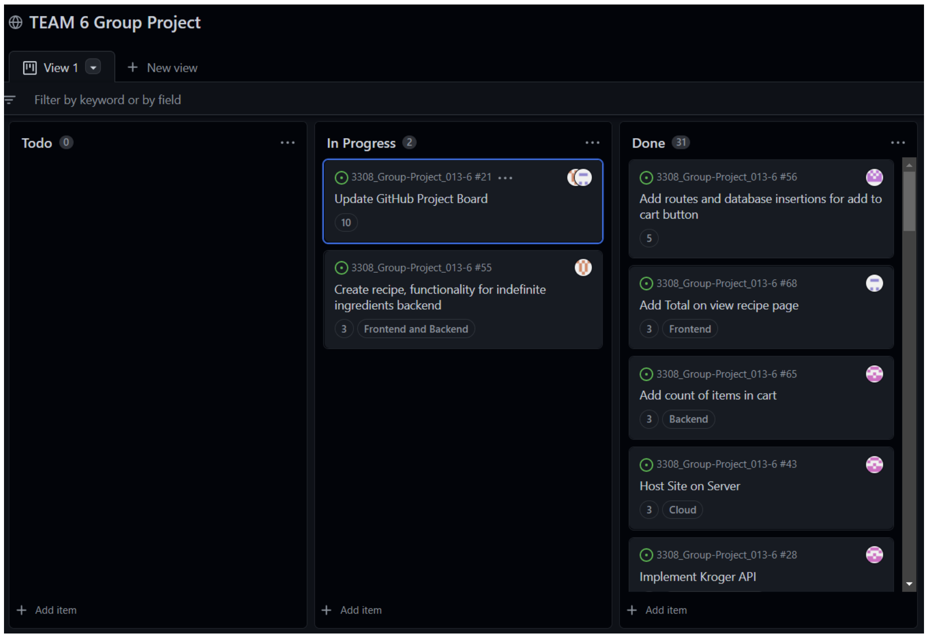
Developers:

Nathan Keyt
Niki Hellmers
Maxwell Prue
Aieshah Safi
Bennett Fragomeni

Project Description:

Frugal Finder is a website designed with the college student in mind, specifically easing their cooking and eating habits by providing a one stop shop to create and save their favorite recipes, while also being able to see the total cost and ingredients needed for a specific recipe. Frugal Finder allows users to create their own recipes, and add recipes to their cart that other users have created. Once recipes are added to a cart, the website has integration with Kroger grocery stores API to be able to see how much your cart would cost and gives you exact products for each ingredient with the lowest possible price, providing an invaluable tool for poor college students. In this way the website allows college age adults to live healthy, flavorful lives without spending too much on groceries. Deliverable uses 30th street King Soopers in boulder as an example store.

Project Tracker:
https://github.com/users/Bennett-Fragomeni/projects/2



TEAM 6 Group Project

View 1 | New view

Filter by keyword or by field

Todo 0

In Progress 2
- 3308_Group-Project_013-6 #21 — Update GitHub Project Board — 10
- 3308_Group-Project_013-6 #55 — Create recipe, functionality for indefinite ingredients backend — 3 — Frontend and Backend

Done 31
- 3308_Group-Project_013-6 #56 — Add routes and database insertions for add to cart button — 5
- 3308_Group-Project_013-6 #68 — Add Total on view recipe page — 3 — Frontend
- 3308_Group-Project_013-6 #65 — Add count of items in cart — 3 — Backend
- 3308_Group-Project_013-6 #43 — Host Site on Server — 3 — Cloud
- 3308_Group-Project_013-6 #28 — Implement Kroger API

Add item

Video:
https://drive.google.com/file/d/1AEAUi4r4pPBk-zvqYvJgIa_JfPKJAzrs/view?usp=share_link

VCS:

https://github.com/Bennett-Fragomeni/3308_Group-Project_013-6

Contributions (less than 100 words):

Nathan Keyt:

Did frontend for login.ejs, register.ejs, recipes.ejs, and view_recipe.ejs. Did backend for view_recipe.ejs and search. Did javascript for indefinite number of inputs on create_recipe.ejs. Implemented Kroger API calls and created getPriceByIngredientName() function utilizing parallel calls to speed up fetching information from API. The function searches an ingredient and finds product with lowest possible price including promos. Did adding and deleting of existing items as a count and totalling cost of all items in cart on cart.ejs. Converted cart backend to getCart() helper function. Did templating for header.ejs, footer.ejs, and message.ejs.

Niki Hellmers:

Designed and set up the majority of the database. Did database setup for users, recipes, ingredients, recipes_to_ingredients, unites, and cart tables. Contributed on adding sample information for the database startup. On the frontend, partly implemented the cart.ejs partial and fully implemented the cart.ejs page. Also implemented the total cost for each recipe on the view_recipe.ejs page for the individual recipe. On the backend, designed and implemented the routes for the functionality for viewing items that were specifically on the users cart, and the ability to add and delete items from the cart of the user.

Maxwell Prue

Focused mostly on API development with index.js and backend integration, involving the initial setup of index.js and API calls such as create_recipe. Set up initial API calls for login, register, etc. Made the API for create_recipe as well as the initial frontend page for create_recipe. Created an account for Kroger API, and collaborated with Nathan on its implementation. Organized project board, ensuring acceptance criteria was added for each user-story. On the bookkeeping side of things, dealt with the initial time conflicts with the meetings with TA, communicated deadlines with team members, and contributed heavily to the presentation slides.

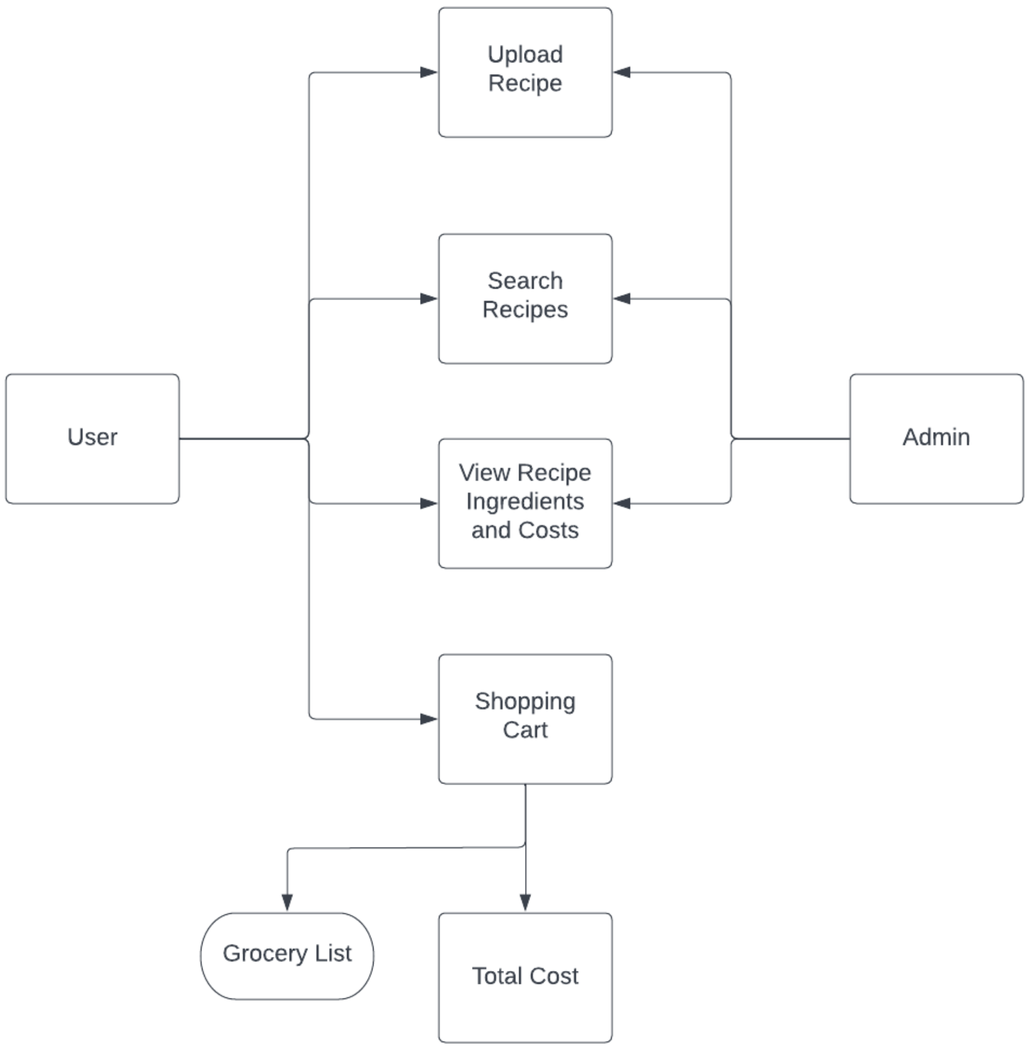
Aieshah Safi:

Focused specifically on all frontend stuff. Worked on register.ejs and login.ejs when it came to how it looked. Focused mainly on the homepage. Added images and description of what our software is. Worked on the add to cart button on pages and focused on designing the view recipes page. Worked on creating a description for the create recipes page and making it more presentable. Created FrugalFinders logo and created Figma to create a visual representation of what we wanted our website to look like. Organized github project board when needed.

Bennett Fragomeni:

Back end Development with some front end integration. Worked on authentication for users on the back end to restrict access to certain areas of the website when they are not logged in. Worked on the front end integration and back end development of a menu, including the ejs page for the menu. Added message passing to the back end. Worked on the integration of the cart, allowing users to add and remove things from the cart as well as view what is in their cart on a drop down menu. Fixed problems I came across when trying to use the website.

Use Case Diagram:



Test Results:

Tested with CU Boulder student who had limited knowledge of our application before hand. Gave abstract of what our application is designed to do and let him test intuitively from there on out. Onboarding was quick, registering and logging in to an account went seamlessly for him and he was able to access our application quickly. Intuitively knew how to view all recipes. First instinct was to add to cart rather than click to view a recipe. Creating a recipe went relatively smoothly, but didn’t instinctively add image so hit a small block when it required him to do so. He created simple recipes to test, such as nutella or baked beans which only consisted of one or two ingredients. Creating recipe functionality worked properly on backend and recipes were added. When viewing recipe information seemed to be displayed relatively intuitively, but user was especially intrigued when he clicked view cart and saw that the products corresponded to the general ingredient name he put in. Nutella ingredient name improperly returned a small nutella snack package instead of a jar, so that was a small issue with our api calls. When inputting multiple quantities of an ingredient, products don't scale as well. This is because the number of cases needed to be handled for unit conversion would be far too vast to cover in this series of sprints, so this is a feature to be considered for future implementation. Overall testing process went smoothly, with a few bugs and unintuitive features which we will cover potential solutions for in the following list.

Changes/Fixes to consider after testing:

- Make a more intuitive button design for viewing a recipe, such as having view recipe text instead of just div highlighting
- Implement more extensive or potential substitution checks if the API doesn't return the type of ingredients a user wants, see Nutella example
- Create default image for no recipe image being inputted, such as a stock image, would have to consider if ease of recipe creation or if website visuals are more important.
- Handle unit conversion and take quantity into consideration for api and pricing calls, this would be ideal for future implementation, but was infeasible within time limit (i.e. vague units such as slices or pinches would have to be converted to oz for a specific ingredient to find how many of a product is needed)
- General styling improvements could be implemented with more time

Deployment:

The following are instructions to run this application independently.

On local machine

- Navigate to ./src
- Install and open Docker daemon
- Ensure port 3000:3000 is exposed on docker-compose.yaml
- Run "docker-compose up" on terminal

On csci3308.int.colorado.edu

- SSH into csci3308.int.colorado.edu using "ssh @csci3308.int.colorado.edu" in terminal
- Install copy of rootless docker using "dockerd-rootless-setuptool.sh install" in terminal
- Clone local repository using "cd" in terminal
- Ensure permissions are allowed using "chmod 777 ." and "chmod 766 package-lock.json"
- Run "docker-compose up -d" in root

- Run "docker-compose ps" to find the exposed port. This will provide the port used to navigate to the public website given by http://csci3308.int.colorado.edu:<port>
- Here you can access the hosted application.