

MealTime

Team 10 Design Document

Evan Klein, Peter Kfoury, Patrick Sullivan, Sam Richardson, Nick Franz, Logan Stout

Purpose

Leading a healthy life can be challenging. Busy schedules, complex nutritional needs, and budgetary restrictions make it difficult to lead a healthy lifestyle. Our project aims to solve the issue of healthy living by taking those factors into account and providing an all-in-one solution to help our users lead happy, healthy lives.

Functional Requirements

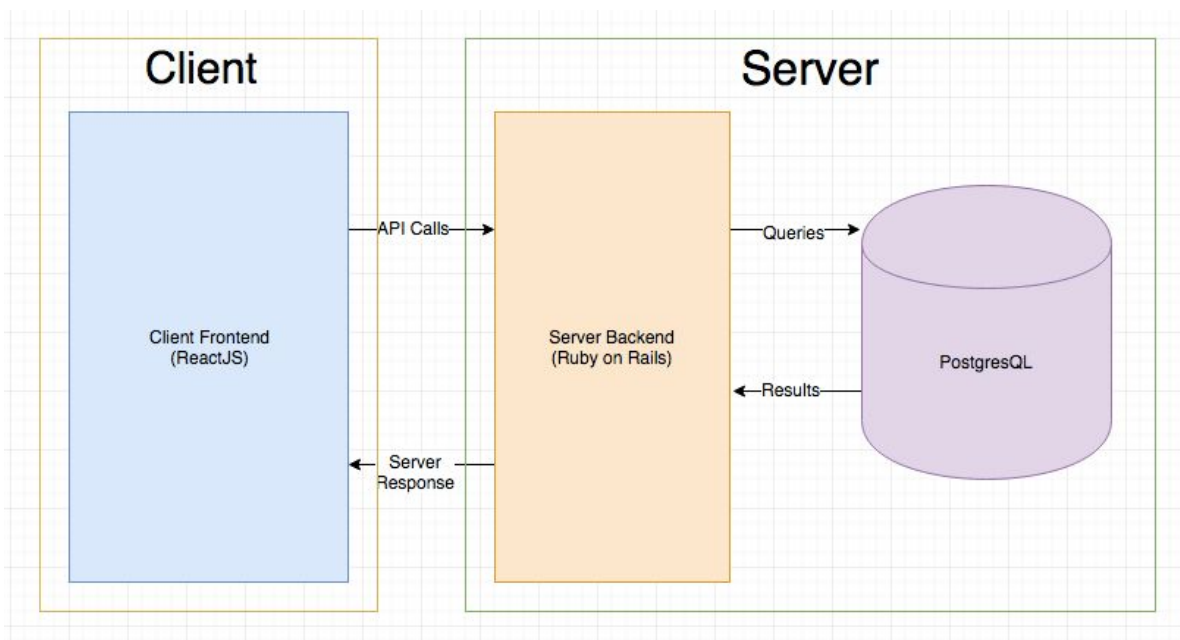
1. Creating an account/sign in
 - a. New users will register with their email, a username, and a password to create an account
 - b. This account will have full usability of the application
 - c. Existing accounts will be able to sign in on the landing page to access their own content that is linked with their own accounts
2. Creating a meal plan
 - a. Any user that is registered can create a meal plan for themselves based off of dietary restrictions.
 - b. Any meal plan that is created will be able to be rated and used by other registered users whose requirements are similar to the meal plan.
3. Using premade meal plans
 - a. Users will be provided with premade meal plans that fit their desired dietary restrictions.
 - b. Users can personalize premade meal plans by taking out, adding, or exchanging meals.
4. Creating meals
 - a. Users who are registered can create their own meals based off of taste and dietary restrictions.
 - b. Meals created will be able to be used amongst other users to add to their own personal meal plans.
 - c. Meals will be able to be rated on a five star system by any registered user.

- d. Meals can be bookmarked by each user.
- 5. Using information about food items
 - a. When creating a meal plan, the user will have access to nutritional facts about the food items they wish to eat.
- 6. Nutritional Stats
 - a. For each user, based on their desired meal plan, there will be an approximate counter for their nutritional intake based on the nutritional facts that are provided.
- 7. Restaurants Nearby
 - a. Restaurants' locations will be added along with their menus so users can choose based on their dietary and monetary restrictions.

Design Outline

High-Level Design of Architecture

We are using a standard client-server architecture for MealTime. From the user's perspective they will interact with a ReactJS frontend. Operations completed on the front end will be communicated to the backend, a Ruby on Rails system. Ruby will interact with a PostgreSQL database.



Detailed Overview of Architecture

Frontend

Our front end will be written in ReactJS with HTML, CSS, and Bootstrap styling. This will be the interface that the user interacts with, and the actual face of our application. All operations that require transferring or pulling information from the user database will be handled on the backend while a friendly user interface greets the user.

The frontend will also be responsible for calling API's of other various sources such as the Yelp API and the USDA API. These operations are not handled on the backend because they don't involve interaction with the user's data, which is the only information we plan on hosting.

Backend

API

Our backend will consist of a Rails backend, written in Ruby. From the API, our front end will make various calls to endpoints to retrieve data from the database. Our backend will also be

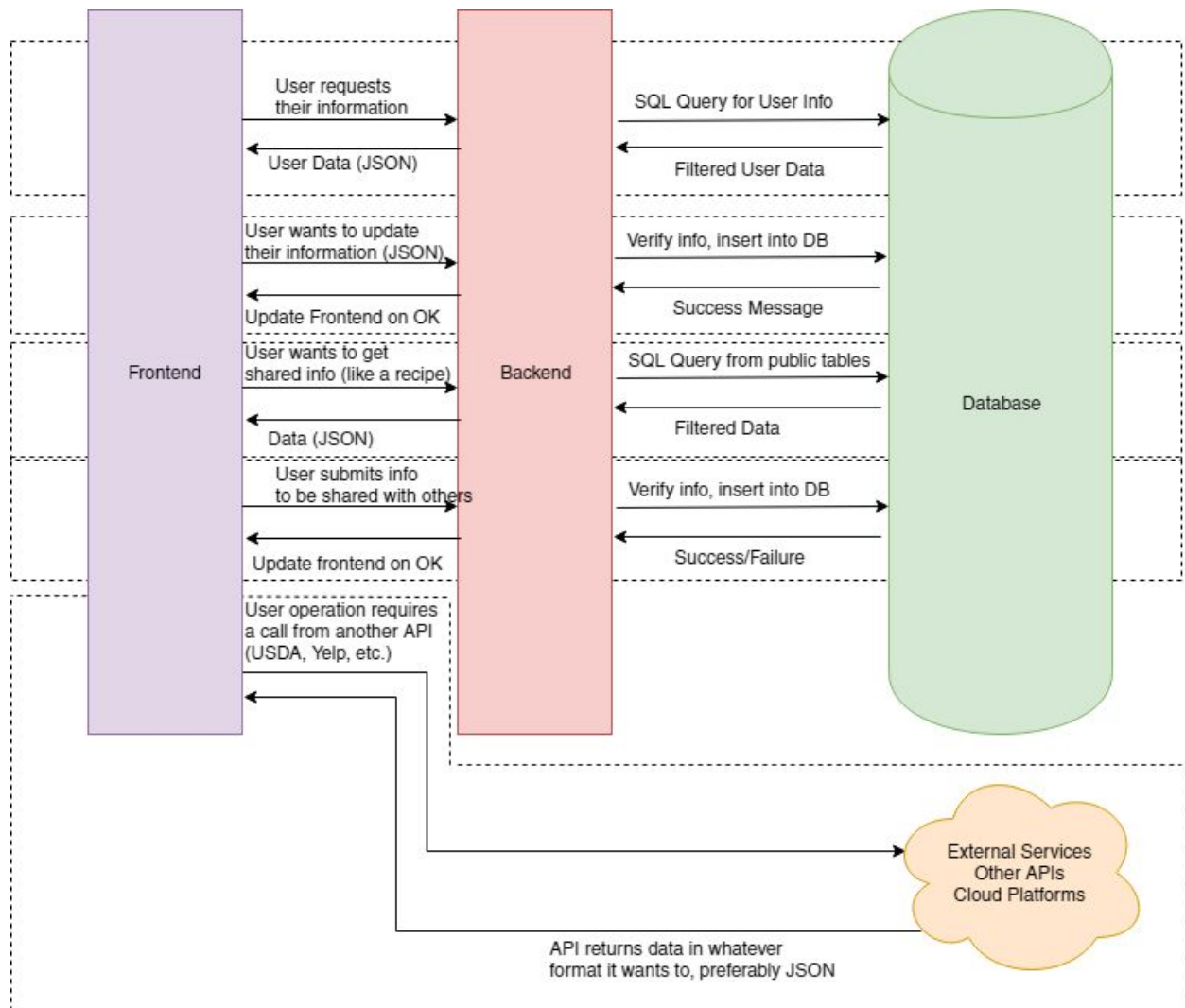
Database

The purpose of the data is to host strictly user information. We will be implementing a PostgreSQL database to store a variety of user information, including their budgetary and nutritional information.

Authorization and Security

Considering we are hosting user's sensitive data, especially health information, security in our system is a high priority. Because of this, there will be validation for identity on the backend that will adhere to Rails's security protocols. Users will not be able to view any other user data, except for features where data can be shared among users (such as shareable recipes or meal plans).

Sequence of Events for System



Design Issues

Functional Issues

1. Should users need to have an account and be logged in to be able to use our service?
 - Option 1: Yes, they must have an account and be logged in to be able to use our services.
 - Option 2: No, we allow users to have full access and look up recipes and find restaurants based solely on their location, without them creating an account.
 - Option 3: No, but we allow guests to have limited access, mainly just general lookup of recipes.
 - Choice: Option 1
 - Discussion: We ultimately decided to go with option 1, mainly because we agreed that because our app is more personalized for a certain person's financial budget and health restrictions, if any, it's better to keep a personalized account that we can better match users with recipes or restaurants that fit their needs.
2. What are all the information we would be asking of from the user to create an account?
 - Option 1: Email, username, password
 - Option 2: Email, password
 - Option 3: Email, username, first and last name, phone number, location, password
 - Choice: Option 1
 - Discussion: We decided option 3 was too much information necessary for our app, which would just be extra data that we have to store, option 2 was too simple and didn't allow anonymity if someone posted a recipe, and option 1 was a good balance, while still allowing some slight anonymity when posting new recipes with the username usage.

3. How should we display the restaurants around the user's current location?

- Option 1: As a list format, sorted to the users preference (rating, price, etc)
- Option 2: displayed on a map, with pinpoints showing each restaurant
- Choice: Option 1
- Discussion: We chose option 1 because first and foremost it would be much easier to implement, and we would not have to find another API and somehow show the restaurants' specific location on a map. We looked into using Google Places API and decided it would extend the amount of work we would have to do considerably, and have looked at other developers' opinions of Google Places API and discovered most others said it was bothersome to work with. Instead, with a list, we could include small information like whether or not a restaurant has certain foods that include dietary restrictions, average pricing, how far it is from the user, and have a simple scrolling function. We can use the Yelp API to acquire that information.

4. How do we want to allow users to interact with user created recipes?

- Option 1: No interaction between users
- Option 2: Commenting
- Option 3: Simple voting system
- Option 4: Commenting and voting system
- Choice: Option 4.
- Discussion: We believed that at minimum we should do option 3, so there's some kind of feedback the users can give to each other, but ultimately decided on option 4, so that when people comment on user created recipes, they could give advice or tips on how to make the recipe better, possible substitutions for ingredients, etc. We would also still be allowing a simple and quick way to show approval or disapproval of a recipe with an upvote/downvote system that can be implemented with a counter and two buttons.

5. What is the minimal amount of information needed to create a new user made recipe?
- Option 1: Just the ingredients, name of recipe, and instructions to cook
 - Option 2: Ingredients, name of recipe, instructions to cook, and time to cook
 - Option 3: Ingredients, name of recipe, instructions to cook, time to cook, difficulty
 - Option 4: Ingredients, name of recipe, instructions to cook, time to cook, preparation time, and difficulty
 - Choice: Option 2.
 - Discussion: We decided that option 2 would be the best minimum amount of information, because to make food from a recipe, it makes sense that all you would need is ingredients to cook, the name of the recipe so others can search for it, how to cook the ingredients, and how it takes to cook them. We would also allow more information to be shown if the user inputs it, like difficulty to cook, possible ingredient substitutions, and pictures of the meal if we have time to implement that.

Nonfunctional Issues

1. What web service should we use?
- Option 1: Amazon Web Services
 - Option 2: Microsoft Azure
 - Option 3: Heroku
 - Choice: Option 3
 - Discussion: While we looked at implementing Amazon Web Services and Azure servers, we ultimately decided that Heroku was the best web service for a few reasons. To start, Heroku is free, while Amazon Web Services and Azure both cost money. On top of this, Heroku is very easy to use and implement, and members of our team also have had

experience using Heroku, and can give advice throughout the project.

While there may be a performance loss using Heroku in comparison to AWS and Azure, the factors listed led us to the conclusion that Heroku is the best option.

2. What backend language/framework should we use?

- Option 1: PHP
- Option 2: Ruby on Rails
- Option 3: Java
- Choice: Option 2
- Discussion: We chose option 2 because there is a lot of documentation on Ruby on Rails, and some of our developers have experience in it. It is also relatively easy to learn and use, while still allowing complex procedures.

3. What frontend language and framework should we use?

- Option 1: HTML and JavaScript
- Option 2: React and JavaScript
- Choice: Option 2
- Discussion: From the beginning we knew we wanted to implement our frontend using JavaScript due to past experience and ease of use, so from there it was just about deciding which framework we would use. Since some of our team members were more familiar with React, we decided that we would implement a React frontend framework.

4. What database should we use?

- Option 1: MongoDB
- Option 2: MySQL
- Option 3: PostgreSQL
- Choice: Option 3
- Discussion: When we were deciding which database we were going to use, we first had to decide whether or not we wanted to use a relational

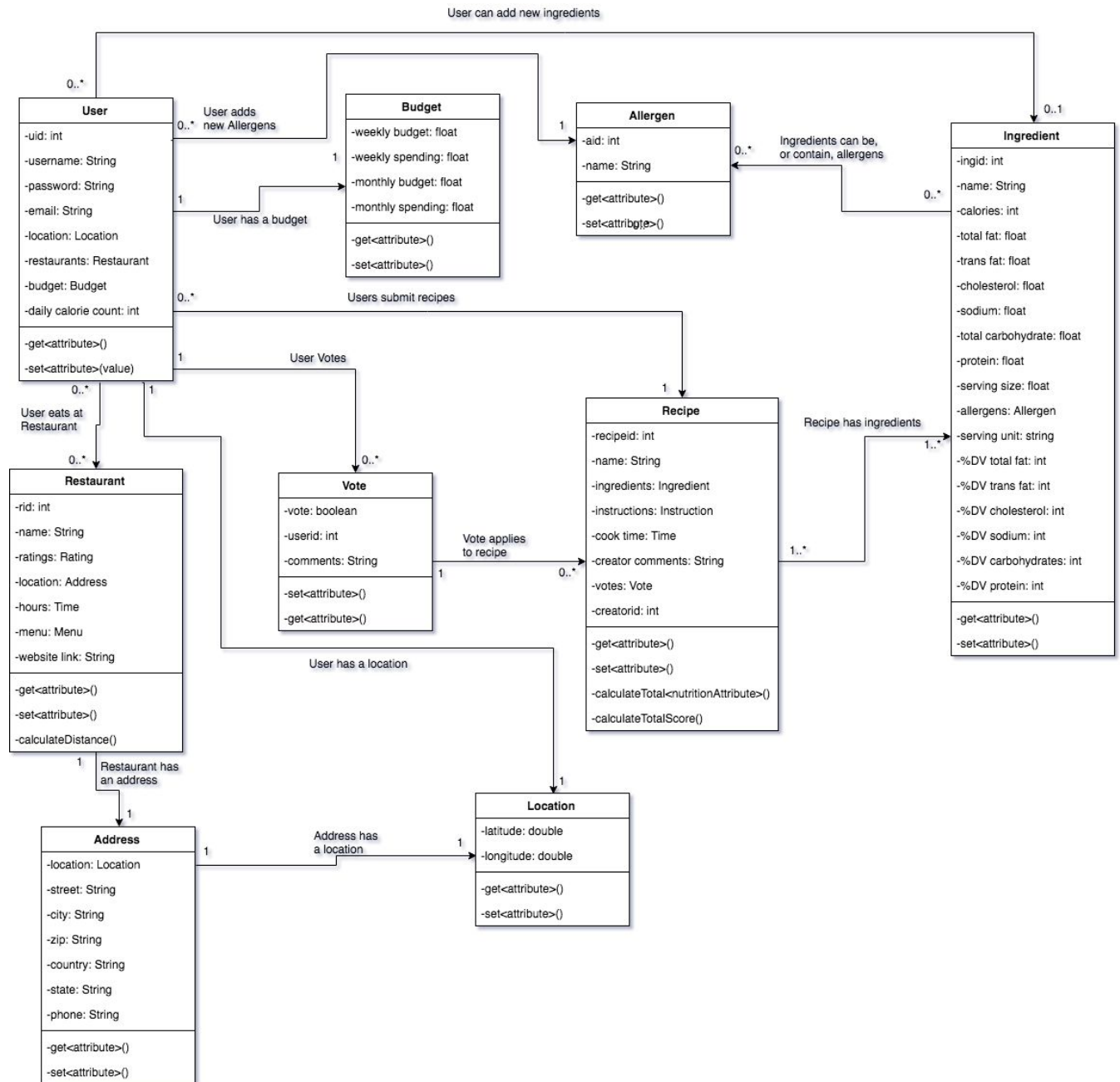
database or a non-relational database. We decided to go with a relational database because we believe that retrieving and storing information will be much easier using a relational database. We decided on PostgreSQL over MySQL because we liked the deeper feature set of PostgreSQL.

5. Which API should we use to acquire information about restaurants?

- Option 1: Zomato API
- Option 2: Yelp API
- Choice: Option 2
- Discussion: While the Zomato API has a deep feature set that would provide almost if not all the information we need about restaurants, we decided that the Yelp API would be the API we used. This is primarily because of the familiarity in general that our team and many users have with Yelp. It also helps that for every feature that Zomato does, Yelp also does.

Design Details

Class Diagram



Class Detail and Interactions

We have provided the general class detail design above. This design is not final but satisfies the current requirements for the desired functionality of MealTime.

1. User
 - a. Basic information on any person using MealTime such as username, password, email, and location
 - b. Other information includes budgeting information, daily caloric intake, and restaurants the user is interested in.
2. Budget
 - a. Created by a user and contains weekly budgeting information for the user.
 - b. Combines weekly budgeting information into monthly information for tracking general spending trends over multiple weeks.
3. Restaurant
 - a. Added by user to a list of potential restaurants to visit.
 - b. This data is pulled from the Yelp API and cannot be altered through mealtime.
4. Address
 - a. Contains normal human readable address information on the restaurant.
 - b. Contains a Location class for the precise location of the restaurant.
 - c. Each Restaurant corresponds to one address. If a restaurant has multiple locations, it will be under multiple restaurant entries.
5. Vote
 - a. Created by a user when they choose to vote on a recipe.
 - b. Contains a boolean variable to track if a user likes (true) or dislikes (false) a recipe.
 - c. This class also allows for the user to provide comments on the recipe as he/she sees fit.
6. Recipe

- a. Created by users and can be seen publicly by any user on MealTime.
- b. Details the ingredients, instructions, cook time, creator comments, and ratings.
- c. This class also totals the nutrition information with all ingredients to give the user an overall nutrition report on the recipe.
- d. Recipes will also have an overall rating calculated by all of the positive and negative votes.

7. Location

- a. Only contains latitude and longitude values.

8. Ingredient

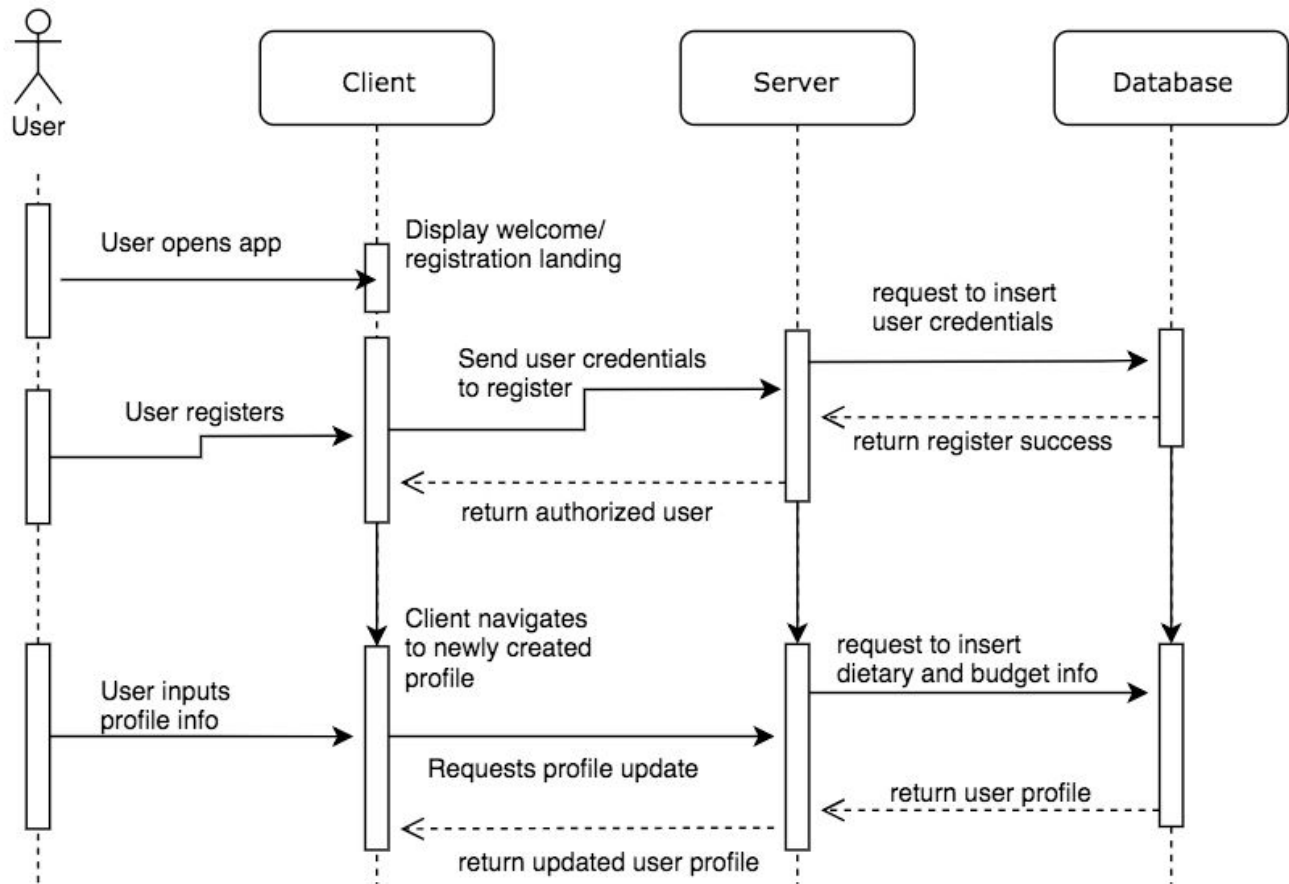
- a. Some are created from first launch but new ingredients can be added by users.
- b. Provides nutritional information on specified ingredient.
- c. Contains a list of allergens associated with ingredient.
- d. This class also includes a serving unit field to allow for more diverse serving measurements (pieces, oz, lbs, etc.).

9. Allergen

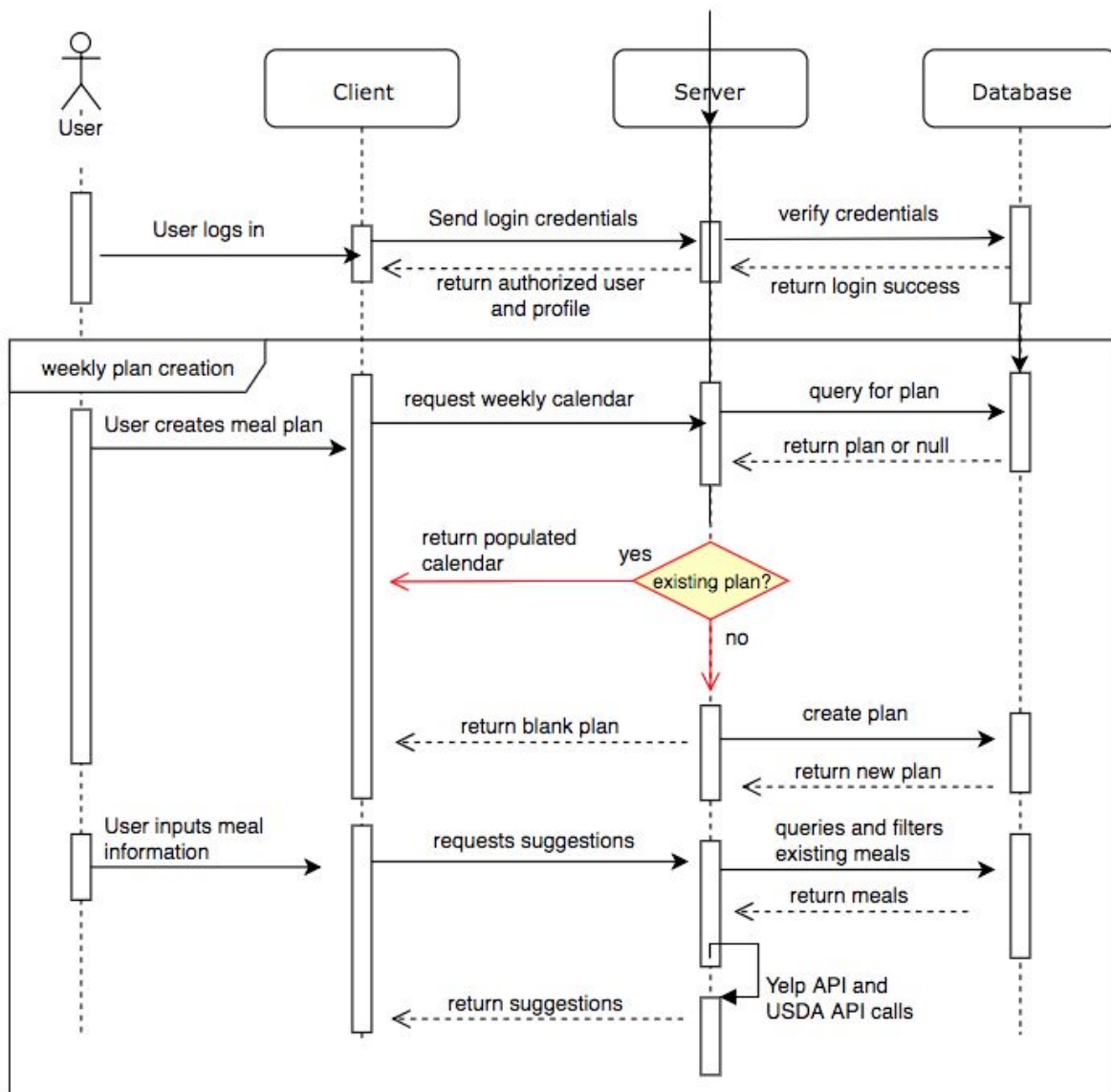
- a. Can be added by users and are not required to be associated with an ingredient to exist.
- b. Provides the allergen name and can be referenced by multiple ingredients.

Sequences of Events

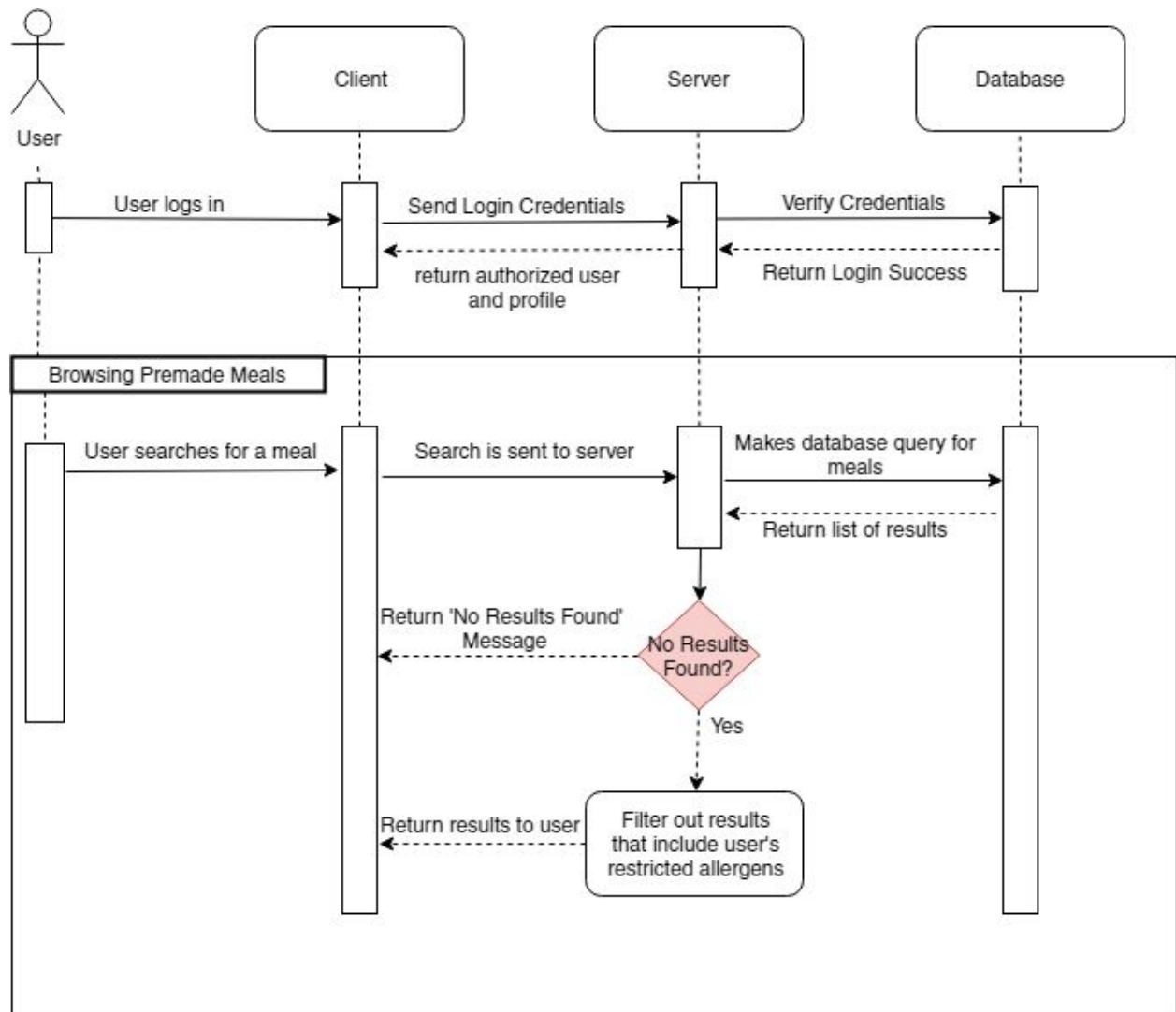
1. User launches app for the first time



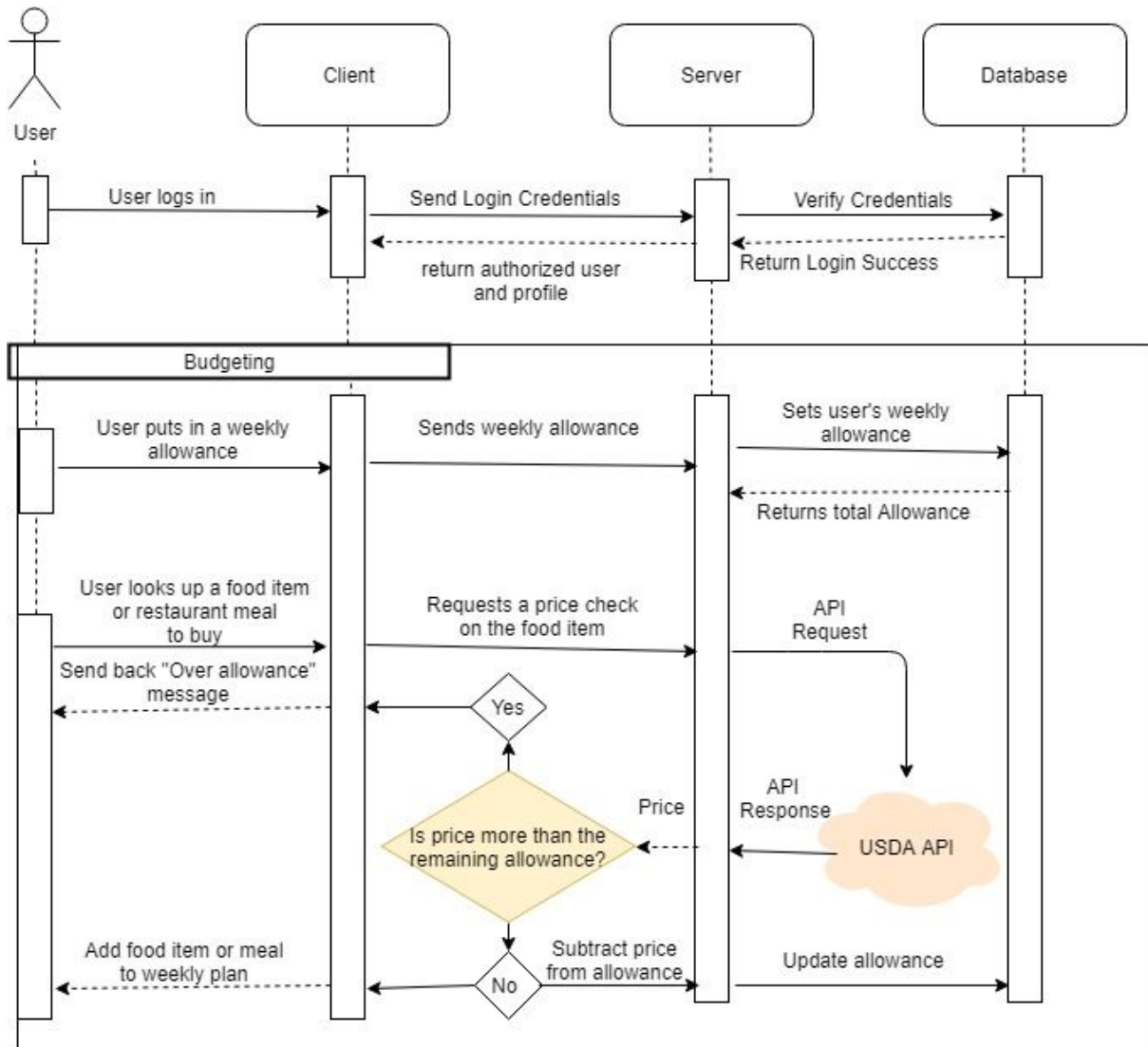
2. User creates meal plan



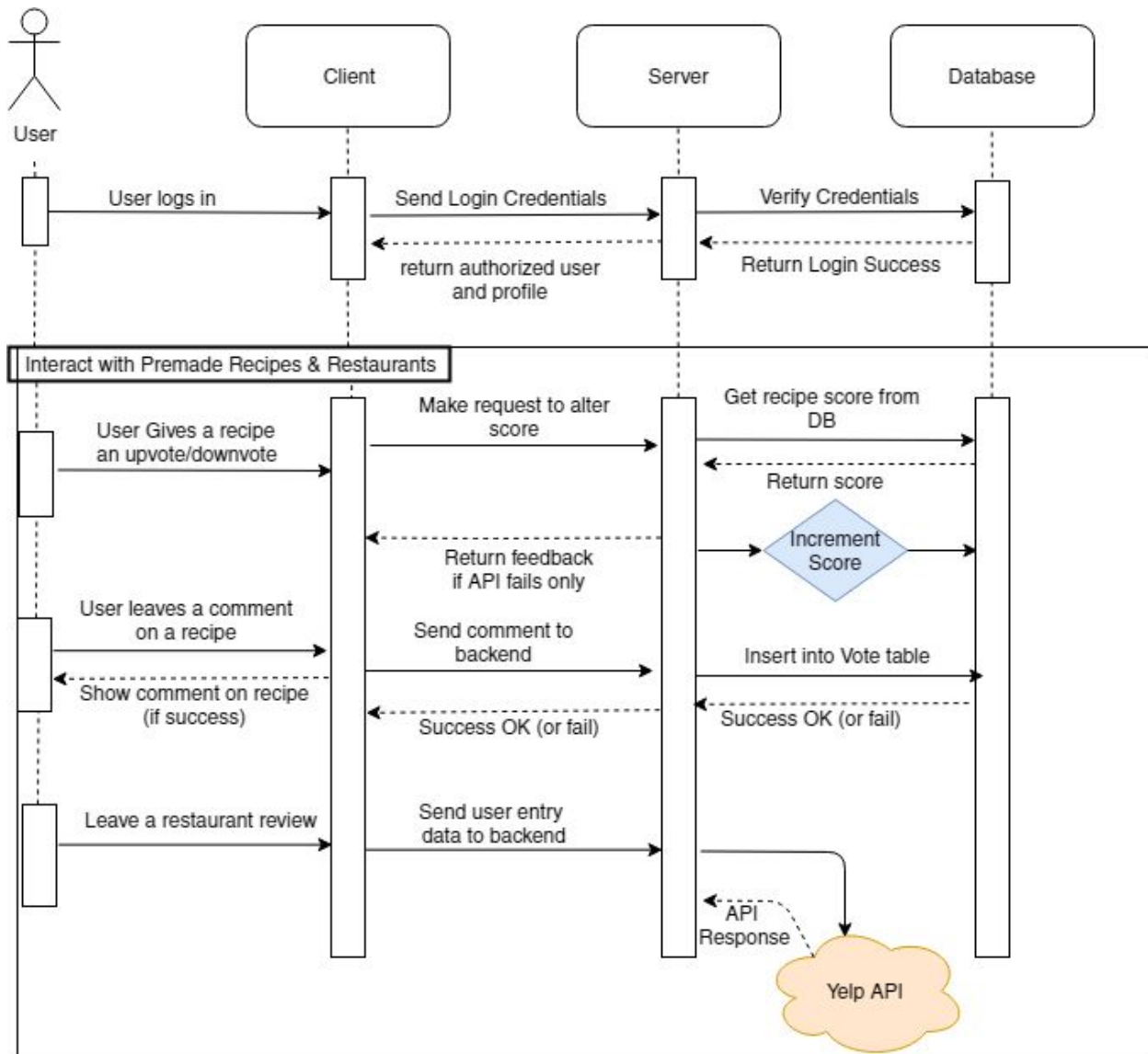
3. User browses premade meals (with dietary restrictions)



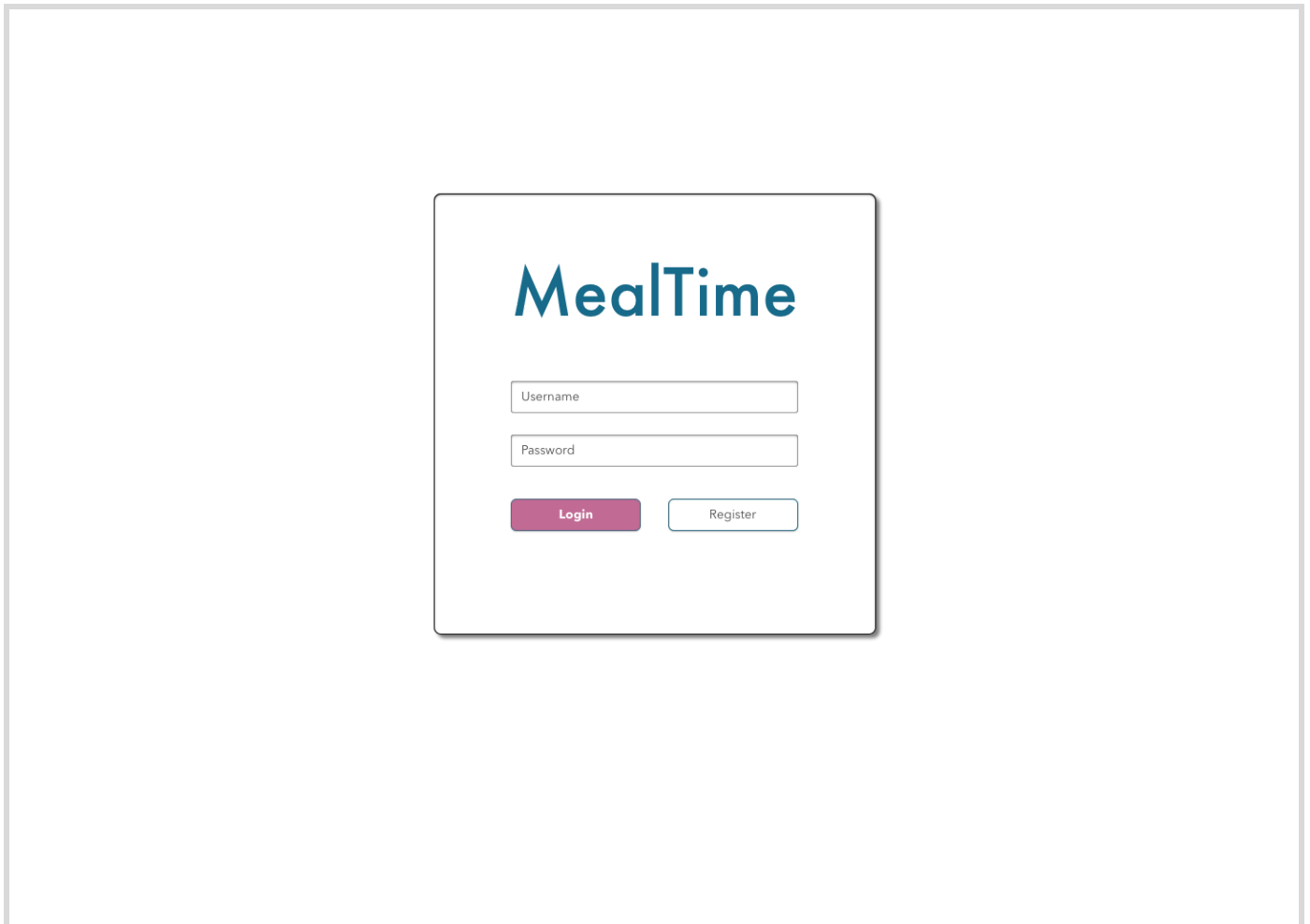
4. Budgeting



5. Various operations (Voting, commenting on recipes, making reviews for restaurants)



UI Mockups



A UI mockup of a desktop login page for 'MealTime'. The page is centered within a large light gray rectangular frame. At the top center is the 'MealTime' logo in a dark blue, sans-serif font. Below the logo are two white input fields with thin gray borders. The first field is labeled 'Username' and the second is labeled 'Password'. Below these fields are two buttons: a solid purple button labeled 'Login' and a white button with a blue border labeled 'Register'.

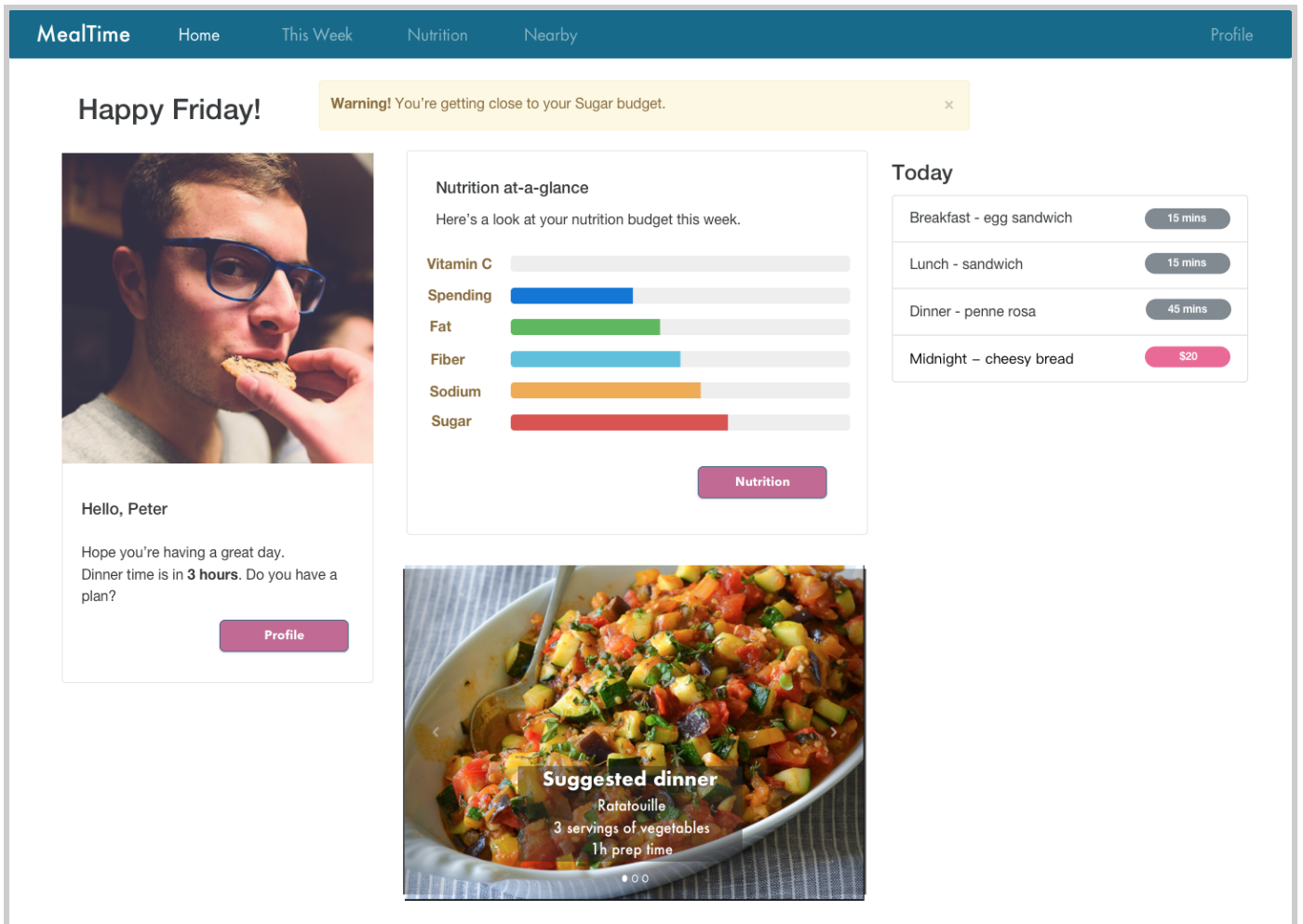
MealTime

Username

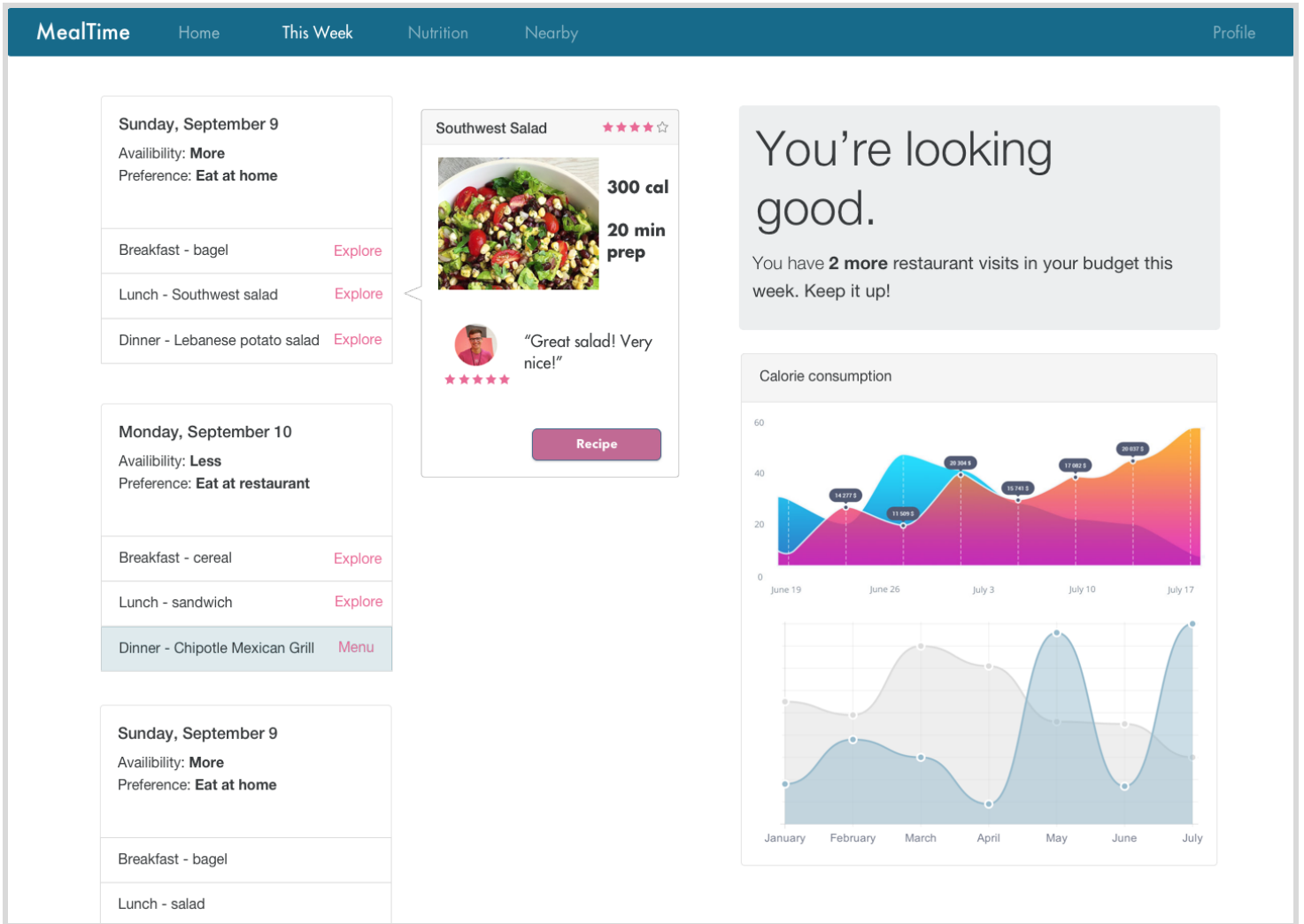
Password

Login Register

Desktop login page



Home - the user sees this page after logging in. It displays at-a-glance information and suggestions.



This Week - the user schedules specific meals for the week here, and can browse more information about popular recipes or orders at restaurants.

MealTime


Home

This Week

Nutrition

Nearby

Profile



Chipotle

Fast-food chain offering Mexican fare, including design-your-own burritos, tacos & bowls.


Popular item - Chicken Burrito Bowl

600 calories

7g fat

600mg sodium

80% MealTime users recommend



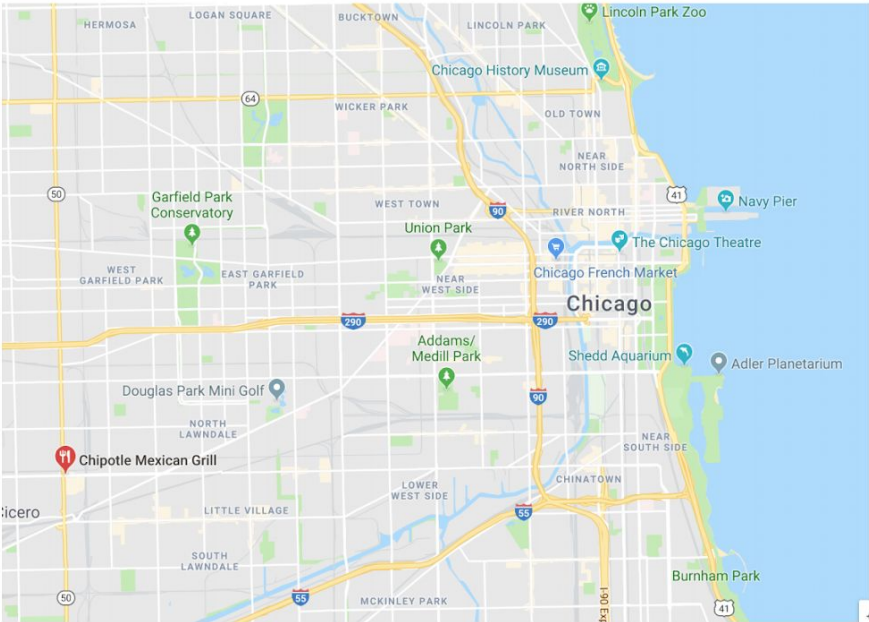
Chipotle

Fast-food chain offering Mexican fare, including design-your-own burritos, tacos & bowls.

Popular item - Chicken Burrito Bowl

600 calories

7g fat



This restaurant is within your monetary budget.

This restaurant has been reported to serve food high in sodium.

Nearby - the user can browse nearby restaurants and make responsible decisions based on nutritional and monetary budget information.