Füdstops

Team 2 Design Document

https://github.com/yyu2002/fudstops

Young Yu, Aaron Kim, Veer Sahani, Reuben Varghese

Navigation

1.0 Purpose

Busy college students don't have a convenient and centralized place to view how busy a dining court or retail restaurant is at certain hours during the day to help determine where to eat (along with knowing what food items are provided at dining courts at that time). Furthermore, students currently do not have the ability to rate food items they consume from dining courts (and retail restaurants), and receive recommendations on where they can find the food that they enjoy (or similar food). They also don't have the ability to easily filter down menu items on the menu to quickly find the type of food they are looking for.

Our web application software, Füdstops, will address these needs by aggregating relevant information so that it is publicly available, giving students the ability to rate their menu items, and creating a recommendation system to help students find dining establishments favorable to their specific dietary needs and preferences. Füdstops will collect data from its users including but not limited to their food preferences, food restrictions, ratings for food items at dining courts, and using such relevant data to provide recommendations on where the user should eat. We will also collect data about the food served at dining courts through web-scraping in order to be able to aggregate menu items and allow users to quickly filter down menu items to what they are looking for. Füdstops collects data on the busyness of dining courts to let students know how busy a dining court typically is at any hour of the day.

To keep it concise, the purpose of our application is to help students make quick and convenient decisions on where they should eat next. Füdstops provides students with a centralized platform to receive personalized recommendations on where to eat on campus based on their dietary restrictions/preferences and ratings of food items, to allow them to more easily find the type of food they are searching for, and to allow them to view the busyness of the locations they want to eat at.

1.1 Functional Requirements

1.1.1 Login, User Profile, General

1. As a user, I would like to log in using my Purdue SSO login, email, or phone number
2. As a user, I would like to be able to reset my password associated with my account if I registered using email or phone number and forgot my password.
3. As a user, I would like to be able to create a username associated with my account
4. As a user, I would like to be able to create a profile picture
5. As a user, I would like to be able to edit my personal information
6. As a user, I would like to be able to access Füdstops on both mobile and desktop devices.
7. As a user, I would like there to be a help feature in order to report a problem.
8. As a user, I would like all my data associated with the app to be stored in my account that I login with

1.1.2 Viewing Active Hours of Dining Courts

9. As a user, I would like to be able to view how busy a dining court is at all hours during the day.

1.1.3 Viewing Menu Items

10. As a user, I would like to be able to view what menu items are available at each dining court on a specific day during breakfast, lunch, and dinner hours.
11. As a user, I would like to have access to the nutrition facts for each food item on the menu.
12. As a user, I would like to be able to view popular menu items at each dining court (algorithm to calculate this TBD).
13. As a user, I would like to see whether a food item will be served on future dates.

1.1.4 Filtering Food Spots

14. As a user, I would like to be able to filter food spots by only those that align with my dietary preferences/restrictions (e.g. vegetarian, soy-free, gluten-free, etc).
15. As a user, I would like to be able to filter dining courts by those that align with dietary preferences/restrictions that I select in a filter menu.
16. As a user, I would like to be able to order dining courts by their busyness at the current hour.
17. As a user, I would like to be able to order dining courts by their name in alphabetical order.

1.1.5 Filtering Menu Items

18. As a user, I would like to be able to filter menu items by only those that align with my dietary preferences/restrictions (e.g. vegetarian, soy-free, gluten-free, etc).
19. As a user, I would like to be able to filter menu items by those that align with particular dietary preferences/restrictions that I select in a filter menu.
20. As a user, I would like to be able to order menu items by their name in alphabetical order.
21. As a user, I would like to be able to filter menu items by relevant aggregation categories (e.g. low-calorie, low-fat, dessert, etc).
22. As a user, I would like to be able to filter menu items by their cuisine.

1.1.6 Rating Menu Items

23. As a user, I would like to be able to rate food items I have consumed.
24. As a user, I would like to vote for commonly clicked on items in the menu so that they appear in dining courts.

1.1.7 Saving Menu Items

25. As a user, I would like to be able to save menu items I enjoyed.
26. As a user, I would like to be able to find recommendations for food similar to menu items I saved.

1.1.8 Dietary Preferences/Restrictions

27. As a user, I would like to be able to select my dietary preferences.
28. As a user, I would like to be able to select my dietary restrictions.

1.1.9 Receiving Recommendations

29. As a user, I would like to receive recommendations for dining courts based on my food preferences/restrictions.
30. As a user, I would like to receive recommendations for dining courts based off of my distance for each dining court.

31. As a user, I would like to have a tab I can click on to generate recommendations on where I should eat for my next meal and the item(s) I should eat from that spot (breakfast, dinner, lunch)
32. As a user, I would like to have a tab I can click on to generate recommendations on where I should eat for my next meal (breakfast, dinner, lunch)
33. As a user, I would like to be able to sort my recommendations by rating, relevance, and distance.

1.1.10 Viewing Food Spot Information

34. As a user, I would like to be able to view the opening and closing hours of a dining court
35. As a user, I would like to be able to view the contact info of a dining court
36. As a user, I would like to be able to get the directions/address to a dining court
37. As a user, I would like to view when breakfast, lunch, and dinner hours are at each dining court

1.1.11 Searching for menu items and food spots

38. As a user, I would like to be able to search for a term, and have all food items matching that term be displayed, as well as the dining court its being served in, the time of day, and on which day

1.1.12 Notifications

39. As a user, I would like to receive text messages (if I opt in) when menu items I enjoyed are offered soon in a dining court
40. As a user, I would like to receive text messages (if I opt in) of recommendations of menu items that I may enjoy based on what I like and my dietary preferences/restrictions.

1.1.13 Retail (if time allows)

41. As a user, I would like to be able to have a separate tab for retail restaurants
42. As a user, I would like to be able to order retail restaurants by their distances from me
43. As a user, I would like to be able to filter retail restaurants by their busyness at the current hour
44. As a user, I would like to be able to view how busy a retail restaurant is at all hours during the day.
45. As a user, I would like to be able to filter retail restaurants by whether they contain foods following my dietary preferences/restrictions
46. As a user, I would like to see an about section for each retail restaurant
47. As a user, I would like to get directions or the address to the restaurant
48. As a user, I would like to see the hours of the restaurant
49. As a user, I would like to be able to view the menu of the retail restaurant

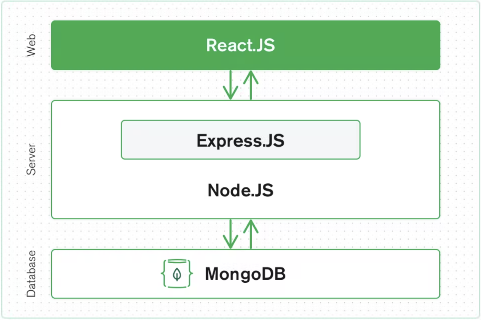
50. As a user, I would like to have access to the nutrition facts for each food item on the menu.
51. As a user, I would like to be able to see when meal swipes are accepted at the retail restaurant during which hours on which days
52. As a user, I would like to be able to view popular menu items at each retail restaurant (algorithm to calculate this TBD).
53. As a user, I would like to be able to filter retail restaurants by only those that align with my dietary preferences/restrictions (e.g. vegetarian, soy-free, gluten-free, etc).
54. As a user, I would like to be able to retail restaurants by those that align with dietary preferences/restrictions that I select in a filter menu.
55. As a user, I would like to be able to order retail restaurants by their name in alphabetical order.
56. As a user, I would like to be able to filter retail restaurants menu items by only those that align with my dietary preferences/restrictions (e.g. vegetarian, soy-free, gluten-free, etc).
57. As a user, I would like to be able to filter retail restaurants menu items by those that align with dietary preferences/restrictions that I select in a filter menu.
58. As a user, I would like to be able to order retail restaurants menu items by their name in alphabetical order.
59. As a user, I would like to be able to filter retail restaurants menu items by relevant aggregation categories (e.g. low-calorie, low-fat, dessert, etc).
60. As a user, I would like to be able to rate retail restaurants food items I have consumed.
61. As a user, I would like to be able to save retail restaurants menu items I enjoyed.
62. As a user, I would like to be able to find recommendations for food similar to retail restaurants menu items I saved.
63. As a user, I would like to receive recommendations for retail restaurants based on my food preferences/restrictions.
64. As a user, I would like to receive recommendations for retail restaurants based off of my distance for each establishment.
65. As a user, I would like to be able to view the contact info of a retail restaurants
66. As a user, I would like to be able to search for a term, and have all food items matching that term be displayed, as well as the retail restaurant which is serving that item, and its hours on the current day
67. As a user, I would like to be able to filter retail restaurants by their cuisine.

1.2 Non-functional Requirements

1.2.1 Architecture

We plan on completely separating the frontend and backend of our application. This is in the interest of intersystem independence as well as for the ease of developing the software.

We will be using the MERN stack (MongoDB, Express, React, Node.JS) to create our web application, mainly with JavaScript and JSON. The frontend will be made using React, and will receive dining data from the backend with API calls. Also, we will use ChakraUI for access to components to make a clean and well-designed web-app.



Frontend - React & ChakraUI , Web Server - Express & Node, Database - MongoDB

1. As a user, I expect to use a web app where my login information/preferences are saved (database).
2. As a user, I expect the website to be straightforward to use and clean to view (React framework + Chakra UI components).

1.2.2 Performance

A key component of this application is that it should be usable on-the-go so that users may quickly receive information or recommendations for dining establishments. As such, it is imperative that the application be as responsive as possible. The application should be able to handle many different requests at once in order to handle spikes in usage during peak dining times.

1. As a user, I expect a response time within 400 milliseconds during idle dining periods.
2. As a user, I expect the web app to have a response time within 600 milliseconds during peak dining periods (12pm - 1pm, 6pm - 7pm).
3. As a user, I expect the web app to be accessible 24/7, with a max down time of 3d 15h 39m 29s (99% SLA).

1.2.3 Security

Security is essential in the development of our software because we will be handling potentially confidential/sensitive user information. This includes, but is not limited to, their authentication information and location. Our software will also be utilizing databases for the storage of multiple types of data. So, preventative measures must be taken to prevent any such exploits, which include maintaining a keen awareness of the security of our app when developing, hiding the visibility of database errors on our product, and sanitizing all user input, including login forms, by removing any potentially malicious code elements such as single quotes.

1.2.4 Usability

Because this application is intended for general use by the Purdue student body, it is necessary that it be as accessible for people of varying technical familiarity and computer literacy. The interface must be intuitive and easy to navigate, with important information such as the most convenient or highest-recommended location being easy to spot.

1.2.5 Scalability

Our software will be extendable to different types of food service locations, not just Purdue University Dining Courts but to other universities if they allow the app to be used for and access their residential dining court information and menus. The software will also be applicable to retail locations, made possible by Google's Place API (tentative).

1.2.6 Hosting & Deployment

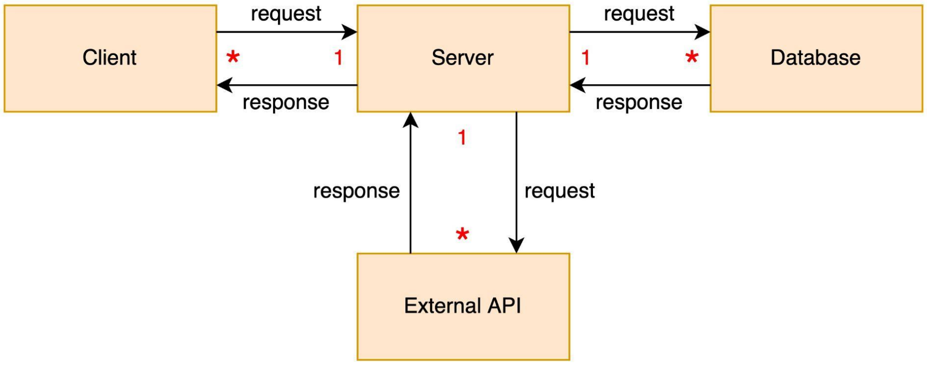
For demonstration purposes we intend to run a local server. MongoDB Atlas will allow us to host MongoDB for our application. The React frontend will be hosted using Netlify and the Node backend will be hosted using Heroku, based on the following reference material: https://niruhan.medium.com/deploying-mern-fullstack-application-on-the-web-for-free-with-netlify-and-heroku-87d888012635. If we deploy the application, we will look into using AWS Web Servers for hosting the webapp for public use.

2.0 Design Outline

2.1 High Level Overview

This project will be a web application which will help its users easily find dining establishments that suit their food preferences, restrictions, schedule and personal taste at dining courts by allowing them to save their favorite foods, set their dietary preferences/restrictions, receive recommendations, view the busyness of dining courts at any hour during the day, and gain access to features that fall under the umbrella of receiving personalized recommendations. The application will also contain features to help the user quickly search for and narrow down menu items to those that suit their personal taste and dietary preferences/restrictions.

The web application will follow the *client-server* model. We will have a server that will handle multiple clients. We will also have database(s) where user information and user specific data like their dietary preferences will be stored. We will also implement a cache for menu item data on the client-side based on the response of the Purdue Dining Menu API to reduce the latency of service such that consecutive requests to the API won't be necessary to retrieve menu data. The client will handle the UI (front-end) while all computation (e.g. for generating recommendations), access to the database, and client requests will all be handled by the server (back-end). The server will also handle all requests to external and third-party APIs. The diagram below highlights each of the high-level components of our system.

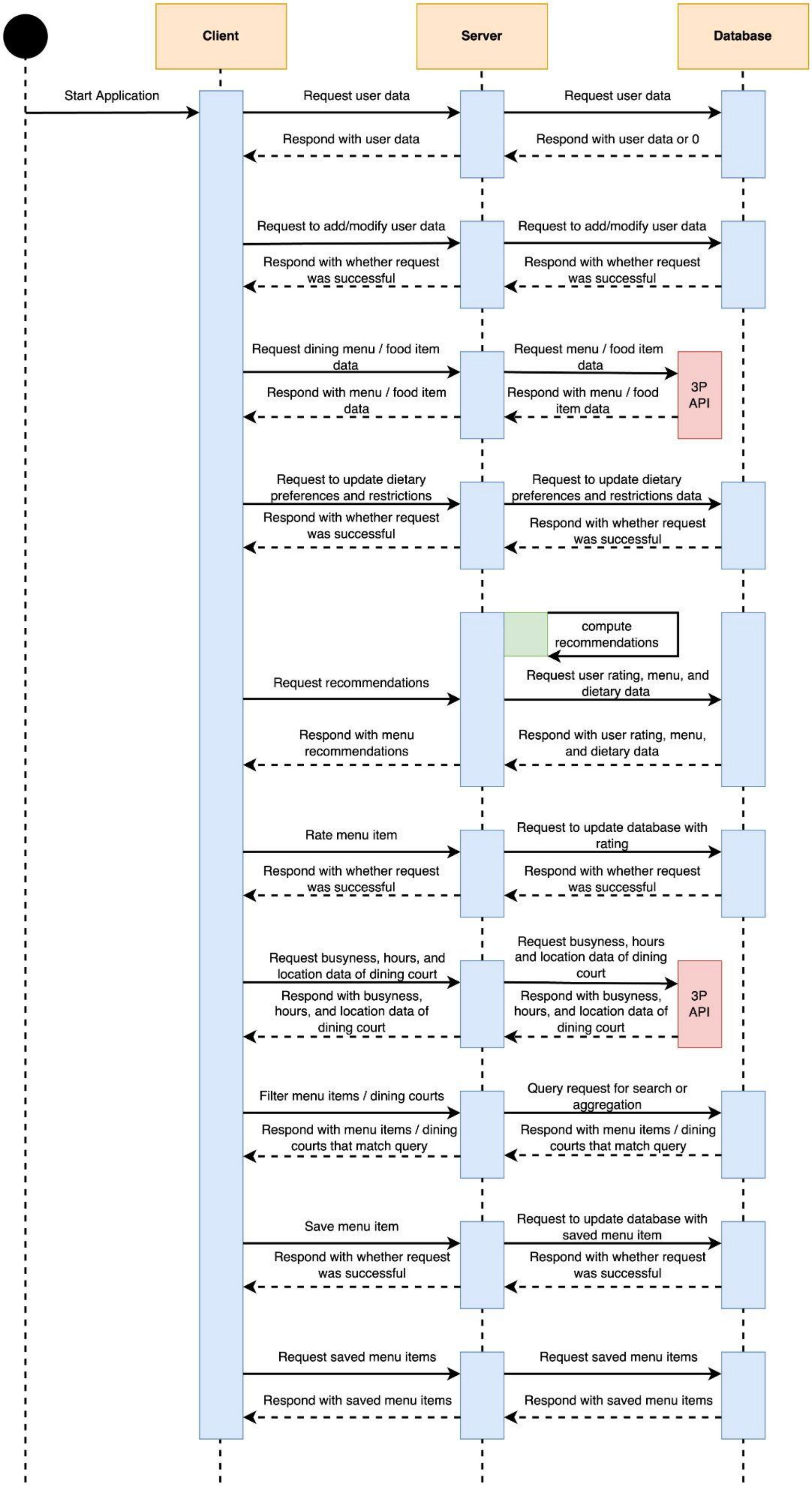


1. Client
    a. The user interacts with the web application through the UI on the client-side.
    b. The client displays all relevant interactive pieces of the UI to the user, including but not limited to menus and menu items for various dining courts, filters, search bars, and buttons.
    c. The client sends requests to the server, handles responses from the server, and handles changes in the UI.
    d. The client caches relevant data in order to reduce latency.
2. Server
    a. Server handles requests from each client, performs all necessary computation, and sends responses to the client.
    b. The server validates requests and sends queries to the database in order to get data, store data, delete data, or update data.
    c. The server handles all requests and responses from external/3P APIs (e.g. Purdue menu API, Google Maps API, etc)
    d. The server caches relevant data from the databases in order to reduce latency
3. Database
    a. The database stores all the data that is used in the application including but not limited to user information, user data, menu data, menu favorites, and busyness data.
    b. The database responds to queries from the server and sends the data back to the server.

2.2 Sequence of Events (High Level)

The following diagram depicts the typical sequence of events that occur when using the application. Everything begins after the user successfully logs in and starts the application. All requests that can be handled by the client without making a request to the server will be handled by the client. Any changes to the database will be done through the client making a request to the server which will *modify the database*, and this is mainly done for security reasons. Requests to *access data from the database* will be done through the client making a request to the server to access. All requests to *access data from 3rd party APIs* will also be handled by the server when a request is made by the client that requires such information. All *computation to determine recommendations* for the user will be done on the server. *Searching and filtering* will be done by the server sending the necessary queries to the database in order to retrieve entries that match.

We will also *cache* information that is likely to be repeatedly needed from the database on both the client and server sides to reduce latency, especially when traffic is high and the load on the server is high (caching is not depicted in the sequence diagram below). An example of this is caching the menu information on the client side so that consecutive requests to view the menu will not require sending a request to the server which will then require send a request to a 3rd party API to access menu data.


Client
Server
Database
Start Application
Request user data
Request user data
Respond with user data
Respond with user data or 0
Request to add/modify user data
Request to add/modify user data
Respond with whether request was successful
Respond with whether request was successful
Request dining menu / food item data
Request menu / food item data
3P API
Respond with menu / food item data
Respond with menu / food item data
Request to update dietary preferences and restrictions
Request to update dietary preferences and restrictions data
Respond with whether request was successful
Respond with whether request was successful
compute recommendations
Request recommendations
Request user rating, menu, and dietary data
Respond with menu recommendations
Respond with user rating, menu, and dietary data
Rate menu item
Request to update database with rating
Respond with whether request was successful
Respond with whether request was successful
Request busyness, hours, and location data of dining court
Request busyness, hours and location data of dining court
3P API
Respond with busyness, hours, and location data of dining court
Respond with busyness, hours, and location data of dining court
Filter menu items / dining courts
Query request for search or aggregation
Respond with menu items / dining courts that match query
Respond with menu items / dining courts that match query
Save menu item
Request to update database with saved menu item
Respond with whether request was successful
Respond with whether request was successful
Request saved menu items
Request saved menu items
Respond with saved menu items
Respond with saved menu items

3.0 Design Issues

3.1 Functional Issues

3.1.1 Do users need to login into our application?

1. Login with Purdue SSO
2. No login required
3. Login with email or phone number

Choice: Login with either Purdue SSO, email, or phone number

Justification: It is necessary for the user to login in to our application in order to use all of the features of this application. If they do not login then there will be no way for us to track things like the user's favorite foods, their dietary preferences/restrictions, and give the user recommendations on where to eat. Therefore, it is necessary for user's to be able to login into the application in order to use all of the features of the web application. Having the ability to login with Purdue SSO, email, or phone number also provides the ability for two-factor authentication, adding an extra layer of security. The user should have flexibility on how they want to login, and thus, after creating an account through either of the three methods, should be able to associate other login information with their account. For example, if they created their account with email, they should also be able to add their phone number to their account and login with it in the future.

3.1.2 How often should we track the users' location data?

1. Always track user location
2. Track user location when they are using the application
3. Never track user's location
4. Users should be able to choose whether their location is tracked (and when)

Choice: Users should be able to choose whether their location is tracked (and when)

Justification: We track the user location data in order to inform them of how far they are from each respective dining court, and this may come into play when designing our recommendation algorithm. For example, if the user is very close to dining court A and it is serving an item they rated 4.5 stars, but very far from dining court B which is serving an item that they rated 5 stars, which item should we recommend? Or if we are recommending multiple items which one should we display ahead of the other? The user's location data is private, and thus they should have control over whether or not this information is tracked. If they want to opt-in to sharing their location data, they will experience better recommendations from the application, but they should always be able to opt-out for personal privacy concerns.

3.1.3 Should users be able to view menu items from past dates?

1. Yes
2. No

Choice: No

Justification: We determined that the core service that we are providing to the customer (the student) is a recommendation engine to help them better decide where to eat in the future. Thus, providing them with information about menu items from the past is irrelevant, although this can be an add-on in the future and A/B tested through conversion rates.

3.1.4 How many days into the future should a user be able to view menus?

1. Only for the current day
2. 1-2 days into the future
3. As far as the data we are able to collect from the menu API

Choice: As far as the data we are able to collect from the Purdue menu API

Justification: Since our goal is to help students decide what to eat around campus in the present and on future days, the customer should have the capability to view what is on the menu on days into the future. To provide the customer with as much relevant information as possible, we should allow them to view the menus into the future as far as the menu API allows. This way they can also plan where they want to eat on a future day.

3.1.5 In what ways should the user be able to narrow down search results to what they are searching for?

1. Search bar
2. Filters
3. None

Choice: Search bar AND filters

Justification: For the convenience of the user, they should be able to search for a particular item they want to eat, and relevant matches should be shown to them. They should also be able to apply filters to the menu items when viewing the menus to conveniently narrow down menu items to those that fit some criteria they have or that fit their dietary preferences/restrictions.

3.1.6 Should a user be able to rate a menu item on a different day if they have already rated it?

1. Yes
2. No

Choice: No

Justification: Each user should only be able to rate a menu item once or else that would give them too much weight towards an item they frequently rate. However, they should be able to change their ratings. There should be some sort of machine learning model applied for fake ratings and to disqualify them though, if there is a user that deliberately gives false ratings to menu items.

3.1.7 How many recommendations should be displayed to the customer?

1. One
2. 5
3. 10
4. 15+

Choice: 10

Justification: This is something that can be experimented with, but starting with 10 recommendations to show to the customer is a good place to start. This gives room for an average of 2 items to be recommended to the user from each of the 5 dining courts. There needs to be a way to rank the recommendations shown to the user though, since we should display more relevant and "better" recommendations to the customer ahead of less relevant ones.

3.1.8 What should be recommended to the customer when there are no "good" recommendations?

1. Nothing
2. The top recommendations from the algorithm
3. Show the recommendations that are above a "score threshold" and also have a button that can redirect them to featured items (highly rated by the general population).

Choice: Show the recommendations that are above a "score threshold" and also have a button that can redirect them to featured items (highly rated by the general population).

Justification: There should be a threshold score for recommendations to the customer, so that recommendations that don't match the customer preferences aren't given to them. In addition, there should be a card after all the recommended menu items that is placed behind all the recommended items which leads the customer to the "featured menu items" (or "popular items") section. We don't want to leave the customer hanging if there aren't too many good recommendations. In the event that none or few recommendations are given to them, they can redirect to the featured menu items and see if there's anything new there that they may enjoy.

3.1.9 What key information should be displayed on the dining court card/buttons

1. Distance
2. The current meal session (breakfast, lunch, dinner)
3. Hours
4. Whether or not the dining court may be busy
5. Whether or not dining court contains a recommended item (and possibly how many)

Choice: All of them

Justification: These are all relevant pieces of information that the user can use to make a decision on whether or not they want to eat at a dining court. We will be tracking the typical busyness of each location using a google maps API, and this can inform the user of whether a dining court may be busy at the current hour or not. This information can be valuable to busy students trying to grab a quick meal. Having an icon indicating whether or not a dining court contains recommended items can be useful to help the user decide where to eat as well. The other information is general important information.

3.1.10 How will recommendations be calculated (and featured / top rated items) when there aren't enough ratings (or very few to none)

1. Use the algorithm without any modifications
2. Make special cases and workflows for the algorithm
3. Don't display that section or say "not enough info collected to provide a recommendation"

Choice: Make special cases and workflows for the algorithm

Justification: We need to consider the situation where none to very few menu items have ratings, or having very few ratings. There should be special cases in the recommendation algorithm that handle these situations. For example, when showing "featured" meal items to the customer and there are no meal items with ratings, should we show zero items to the customer? Or should we show randomly picked items? Or should we collect some sort of less weighted back-up metric to break the tie between all the unrated items? When there are very few menu items with ratings, or none, showing randomly chosen items and collecting some sort of less weighted back-up metric (e.g. how often a menu item is clicked → ones that are more frequently clicked on should be weighted higher) to break the tie between all the unrated items is a good way to solve this issue.

3.1.11 When should user's receive recommendation notifications?

1. Through notifications that will come periodically, before their next meal
2. At specific times during the day that they choose
3. The day before

Choice: Through notifications that will come periodically, before their next meal AND the day before

Justification: The idea of the recommendation system is to send personalized recommendations of where to eat to the user before they decide on where to eat. Thus, before the dining court opens for a specific time frame (e.g. breakfast, lunch, dinner) the user should be notified of recommended meal items at specific dining courts. Additionally, the user should be notified of meal items they may like the day before they are served, so that they can decide where to eat for the upcoming day.

3.2 Non-Functional Issues

3.2.1 What web service should we use?

1. Heroku
2. Google Cloud
3. AWS
4. Azure

Choice: Heroku

Justification: When using the MERN stack, Heroku is typically used for a Node server which is what we are using. Heroku also provides free services which will be perfect for our small scale project. We are mostly focused on designing the application and not worried about the configuration, therefore using Heroku will simplify the process for us and allow us to allocate more time to the functionality of the application since Heroku has features where we would not have to worry about server configuration, network management, and turning the latest version of our database.

3.2.2 What frontend technology should we use for this project?

1. React
2. Vue
3. Angular
4. HTML

Choice: React

Justification: We are planning on using the MERN stack for this project so the typical frontend that is used for this stack is React. React is one of the easiest front end languages to learn and there are many videos and lots of documentation that can help us write code in React. A neat feature in React is that it uses components, which have their own logic and are great due to the code for components being reusable. Also, if any changes are made in a particular component it will not affect other parts of the application. React is also generally seen as a faster language than HTML as it allows us to create small, and more efficient lines of code. The effect of this is that the user interface may be more responsive.

3.2.3 What backend technology should we use for this project?

1. Node.js
2. Java Spring Boot
3. Django
4. PHP

Choice: Node.js

Justification: We are planning on using the MERN stack for this project so the typical backend that is used for this stack is Node.js. Node.js is perfect for our project due to the fact that it makes it easy for us to handle multiple requests made by the client. The fact that Node has single threaded functionality makes it the most suited for real time communication which is essential for our application. Using a backend language like Springboot could be problematic for us because we would have to worry about managing multiple threads, whereas in Node we just have to worry about a single thread.

The two backend languages we had to decide between in the end were Node.js and Java. Node is based on JavaScript, and can be used for both client and server-side which is beneficial for an agile workflow which is the methodology we will be following while developing. Node also used an event looping mechanism which allows servers to respond in a non-blocking way unlike Java and enables higher scalability. The benefit of Java is in its multithreading power, which is something we don't heavily depend on in our software. The other plus of Java is platform independence and portability, as well as robust memory management.

Node performs better than Java typically due to Java's compiler and its garbage collection. In the end, we decided to move forward with Node because of its common use, compatibility, and scalability with the other technologies of MongoDB, Express, and React.

3.2.4 What database should we use?

1. MongoDB
2. MySQL
3. Microsoft SQL
4. Oracle XE

Choice: MongoDB

Justification: MongoDB fits in perfectly with the other components of the project that we are using like Express, React, and Node. MongoDB provides us with a full tier of services at free cost which is exactly what we need. MongoDB also has a full cloud-based platform, flexible document schemas, and it also has powerful querying and analytics. MongoDB also supports search index querying so we do not have to use another service in order to allow students to search for their favorite food items. MongoDB Atlas Search offers a full-text search solution that is also scalable. We are working with XML databases and it does not follow a relational model, thus we would have to use NoSQL. MongoDB is perfect because it supports NoSQL.

3.2.5 In order to access location data, which API should we use?

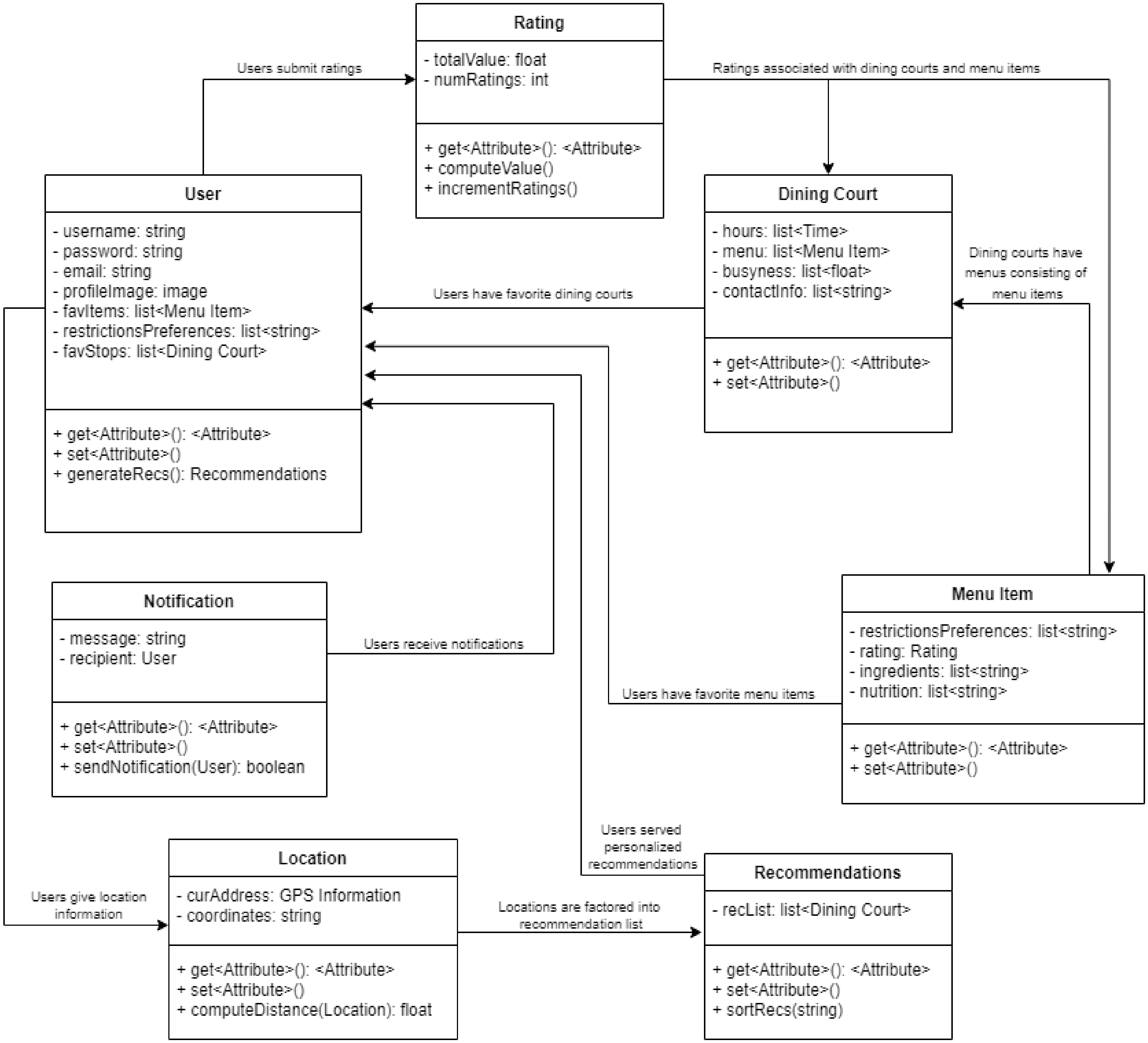
1. Telize GEOIP API
2. Google Maps API

Choice: Google Maps API

Justification: The Google Maps API is the best option for us because it provides multiple API options, and the service gives us the ability to return information about specific places by using HTTP requests. The API is also very neat because it allows us to perform different levels of inquiry. We can use the location maps that can show us either a street view or a satellite view. We can also use geospatial visualization which can make the maps look more visually appealing and also add some useful information to the map. Lastly, Google maps allows us to customize our maps so we can add features like adding colors to the map to highlight specific locations, and remove any map sections. Therefore, the Google Maps API is the best option for our project because it provides us with the most functionality.

4.0 Design Detail

4.1 Class Design


Rating
- totalValue: float
- numRatings: int
+ get<Attribute>(): <Attribute>
+ computeValue()
+ incrementRatings()
Users submit ratings
Ratings associated with dining courts and menu items
User
- username: string
- password: string
- email: string
- profileImage: image
- favItems: list<Menu Item>
- restrictionsPreferences: list<string>
- favStops: list<Dining Court>
+ get<Attribute>(): <Attribute>
+ set<Attribute>()
+ generateRecs(): Recommendations
Dining Court
- hours: list<Time>
- menu: list<Menu Item>
- busyness: list<float>
- contactInfo: list<string>
+ get<Attribute>(): <Attribute>
+ set<Attribute>()
Users have favorite dining courts
Dining courts have menus consisting of menu items
Notification
- message: string
- recipient: User
+ get<Attribute>(): <Attribute>
+ set<Attribute>()
+ sendNotification(User): boolean
Users receive notifications
Menu Item
- restrictionsPreferences: list<string>
- rating: Rating
- ingredients: list<string>
- nutrition: list<string>
+ get<Attribute>(): <Attribute>
+ set<Attribute>()
Users have favorite menu items
Users served personalized recommendations
Location
- curAddress: GPS Information
- coordinates: string
+ get<Attribute>(): <Attribute>
+ set<Attribute>()
+ computeDistance(Location): float
Users give location information
Locations are factored into recommendation list
Recommendations
- recList: list<Dining Court>
+ get<Attribute>(): <Attribute>
+ set<Attribute>()
+ sortRecs(string)

4.2 Description of Classes and the Interactions between them

4.2.1 User

- User will be created when they sign up to use this web application
- Each user will have a username and password that they will use to login
- Each user will be able to update their personal information (name, email, profile picture)
- Each user will have their own list of favorite foods/menu items
- Each user will have their own list of dietary restrictions/preferences
- Each user will have their own list of favorite dining courts
- Each user will be able to filter menu items based on their dietary restrictions/preferences, cuisine, and by different categories of food
- Each user will be able to filter food spots based on their dietary preferences/restrictions, and by how busy a dining court is at a particular moment in time

4.2.2 Location

- Contain the location of the user
- Address of user will be used to find coordinates
- Once we have coordinates, we can compute the distance between the user and the dining court they are trying to go to

4.2.3 Notification

- This class is used to send notifications to users
- User will receive notifications when a menu item they favorited appears on the menu
- User will receive notifications when of recommendations based on their dietary preferences/restrictions

4.2.4 Rating

- Ratings will be created once the user starts giving ratings
- The rating will be computed, and will be put into the database
- Ratings for individual food items will be averaged so that the highest rated food item can appear on the menu again

4.2.5 Recommendations

- Recommendations are given based off the user's dietary preferences/restrictions
- Recommendations are also given based off the users distance from a particular dining court
- Users will be able to sort specific recommendations based on three different categories: rating, relevance, and distance

4.2.6 Dining Court

- Will include key information such as the hours of each dining court
- Will contain the contact information for each dining court
- How busy a dining court is during the hours of operation
- Will contain the menus for each dining court, a list of Menu Item objects

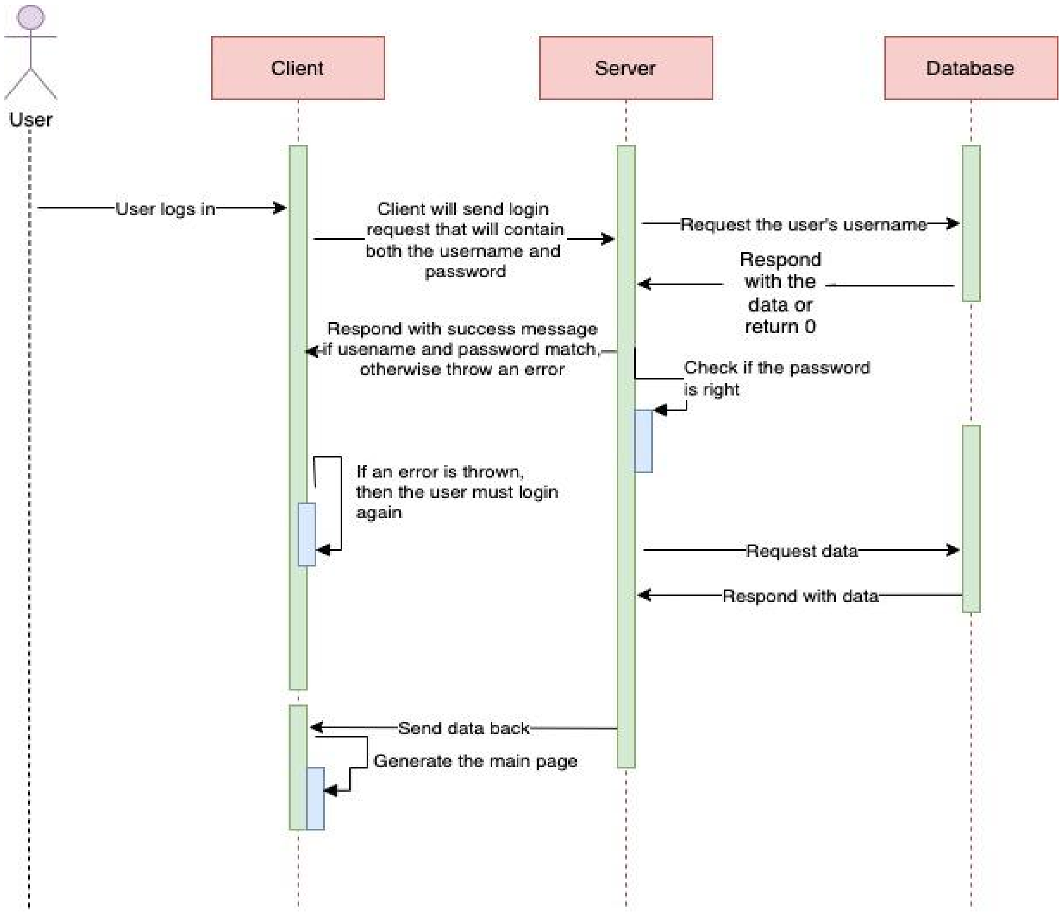
4.2.7 Menu Item

- Each Menu Item has a calculated Rating value
- Contains a list of Nutrition Facts
- Contains a set of preferences/restrictions that the Menu Item falls into
- Contains a list of ingredients

4.3 Sequence Diagrams

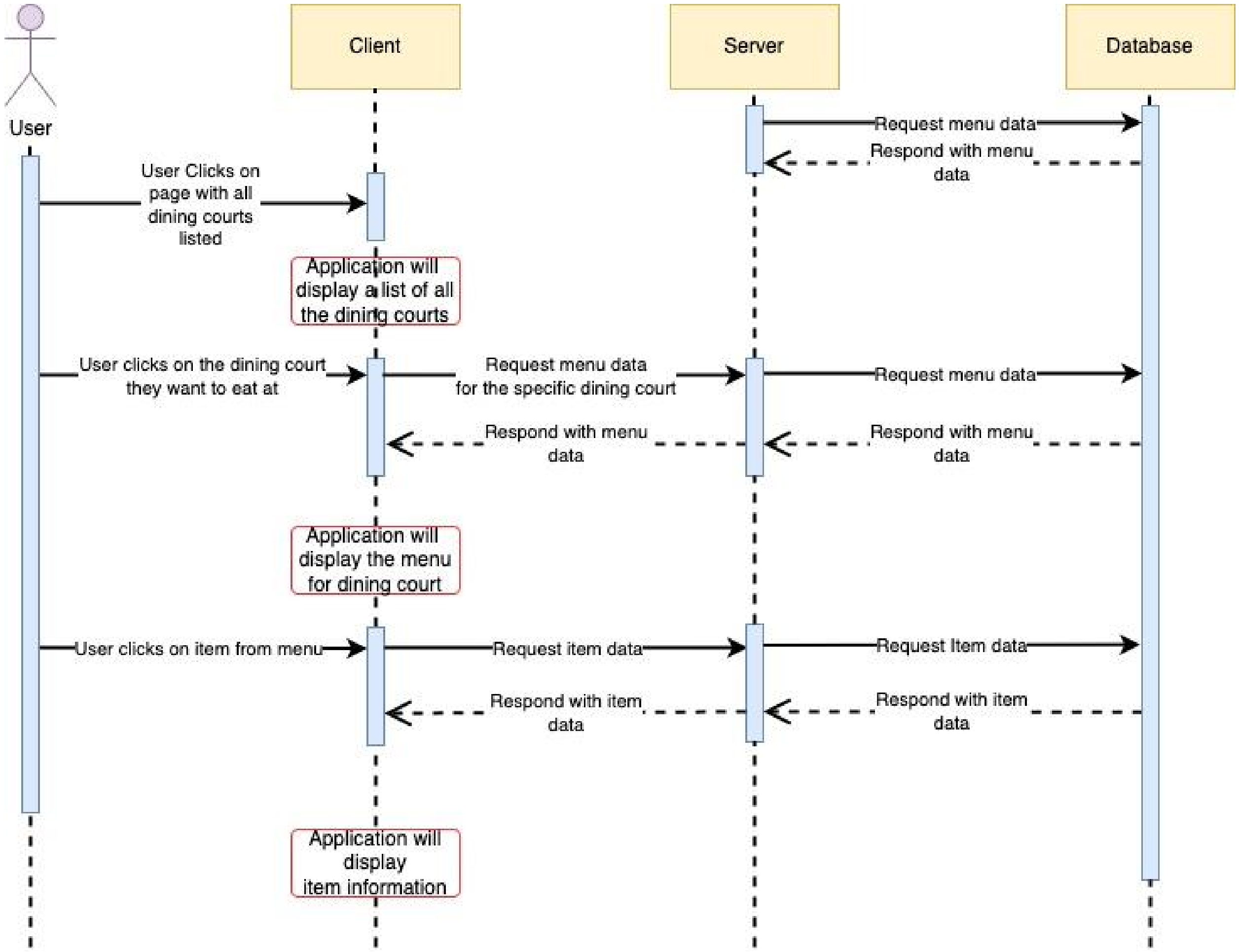
4.3.1 Process for when the user logs in to the application

The following diagram is a typical sequence of events when the user logs in to the application. The diagram portrays the interactions between the user, client, server, and the database. It shows how the user will login, and if an error is thrown the user will be prompted to login again.


User
Client
Server
Database
User logs in
Client will send login request that will contain both the username and password
Request the user's username
Respond with the data or return 0
Respond with success message if usename and password match, otherwise throw an error
Check if the password is right
If an error is thrown, then the user must login again
Request data
Respond with data
Send data back
Generate the main page

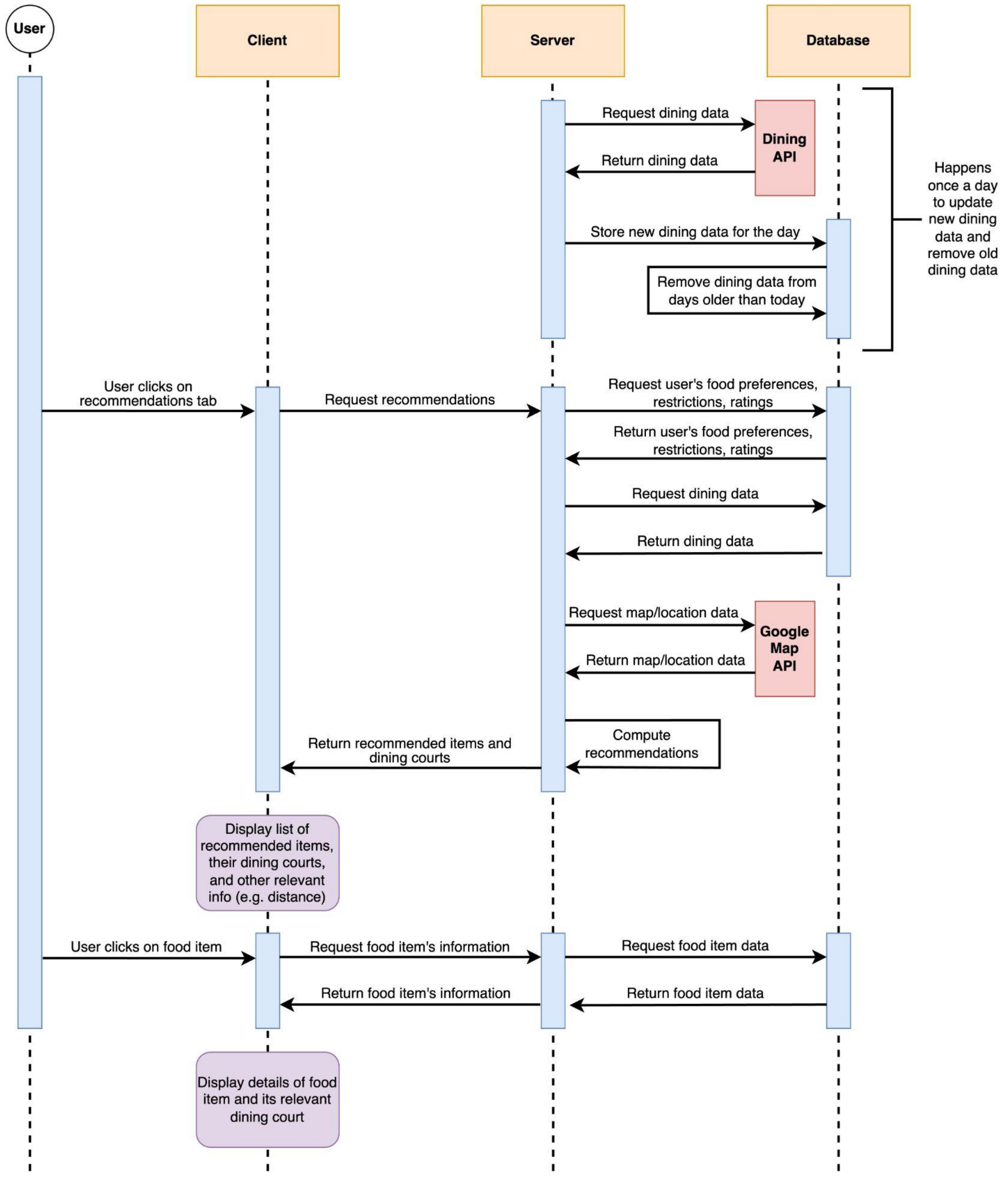
4.3.2 User clicks on dining menu then clicks on one of the menu items

The diagram below portrays the typical sequence of interactions between the user, client, server, and the database when clicking on the menu for a specific dining court. At the top of the sequence diagram, new menu data is added to the database and the old menu data is removed. The diagram shows how the user will interact with the client to get the menu information that they want. They will first start off by going to the page where all the dining courts are displayed. When the user clicks on the dining court, the menu will be retrieved from the database, sent back to the server and the client, and displayed to the user. The same process occurs when the user wants to view specific item information.


Client
Server
Database
User
Request menu data
Respond with menu data
User Clicks on page with all dining courts listed
Application will display a list of all the dining courts
User clicks on the dining court they want to eat at
Request menu data for the specific dining court
Request menu data
Respond with menu data
Respond with menu data
Application will display the menu for dining court
User clicks on item from menu
Request item data
Request Item data
Respond with item data
Respond with item data
Application will display item information

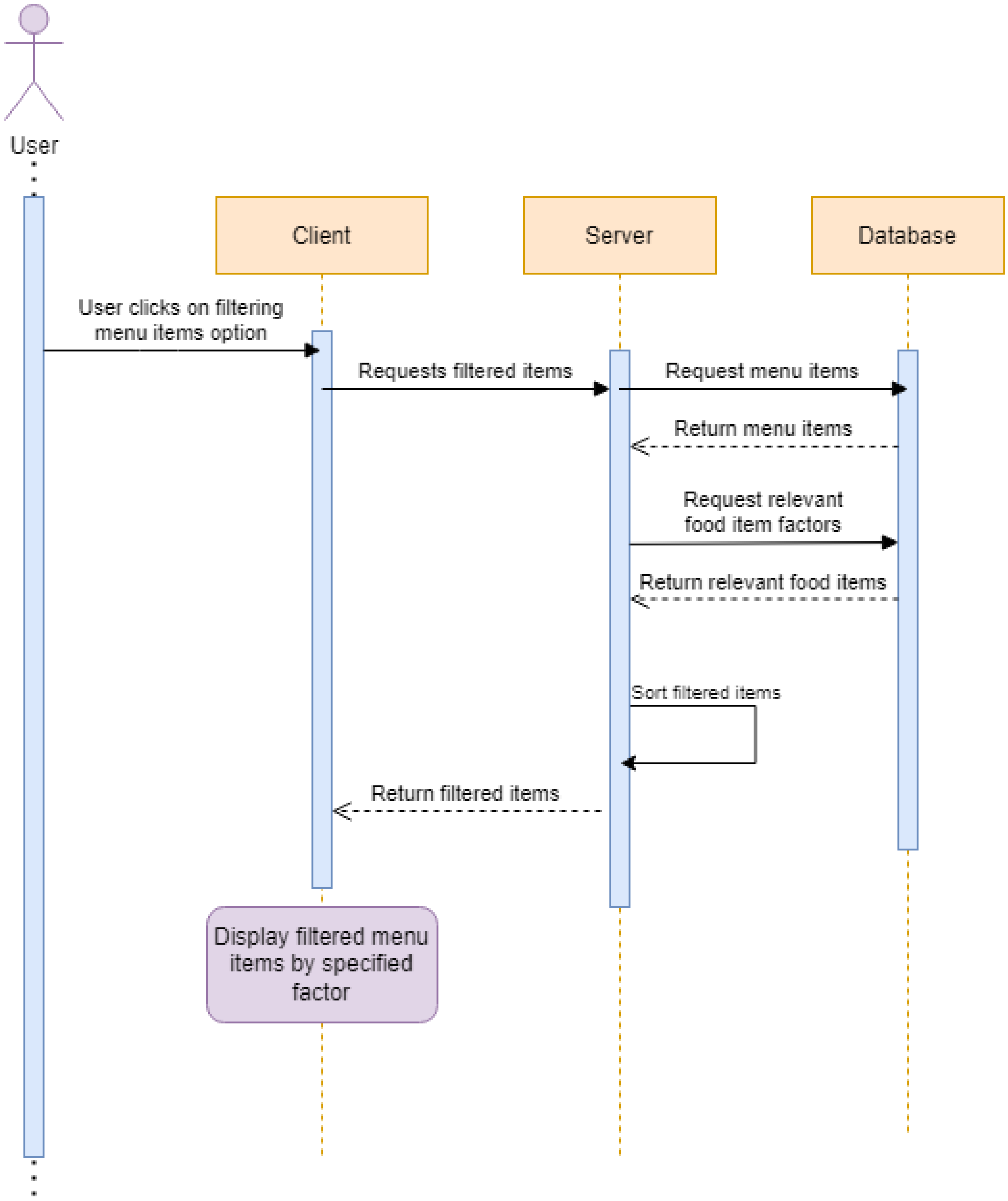
4.3.3 User clicks on the food recommendations tab and clicks on one of the food items

The diagram below portrays the typical sequence of interactions between the user, client, server, database(s), and 3rd party APIs when clicking on the food recommendations tab and clicking on one of the food items listed in the food recommendations page. Depicted at the top of the sequence diagram is a process that occurs once a day where new dining data is added to the appropriate database and old data for old menus is removed. The server takes the user's preferences, ratings, restrictions, location data, and other relevant information to recommend food items to the user. The items will likely be displayed in a carousel. When the user clicks on a food item, the appropriate food data is retrieved from the database and sent back to the server and client to be displayed to the user. We plan on caching food data on the client side to prevent redundant calls to the database for food items already seen by the user.


User
Client
Server
Database
Request dining data
Dining API
Return dining data
Store new dining data for the day
Remove dining data from days older than today
Happens once a day to update new dining data and remove old dining data
User clicks on recommendations tab
Request recommendations
Request user's food preferences, restrictions, ratings
Return user's food preferences, restrictions, ratings
Request dining data
Return dining data
Request map/location data
Google Map API
Return map/location data
Compute recommendations
Return recommended items and dining courts
Display list of recommended items, their dining courts, and other relevant info (e.g. distance)
User clicks on food item
Request food item's information
Request food item data
Return food item's information
Return food item data
Display details of food item and its relevant dining court

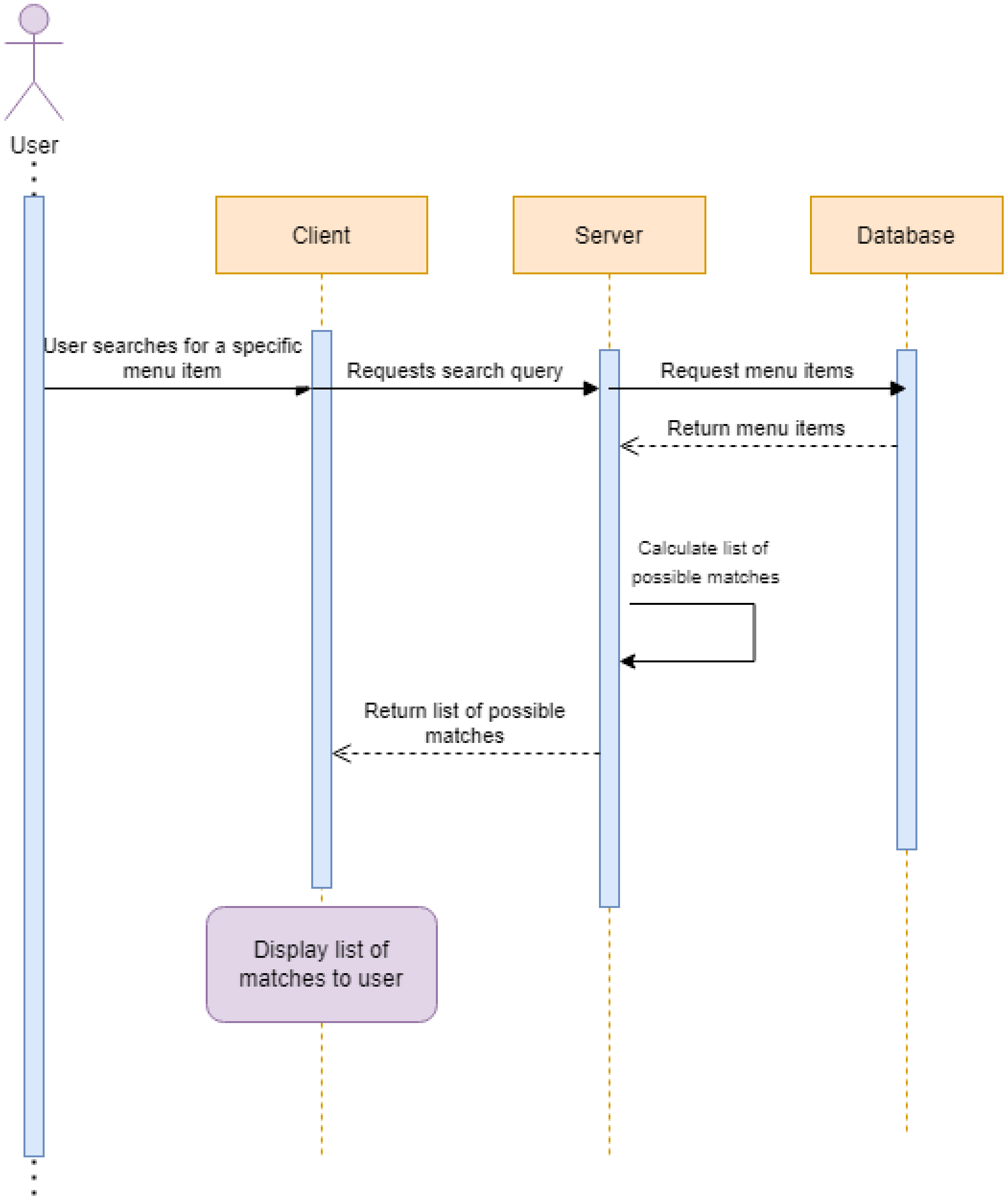
4.3.4 User applies filters to the menu items / sorts the menu items by some factor

User taps on the “filter menu items” option, which sends a request to the server to filter menu items, which then requests the menu items from the database. It also requests the factor, whether it is food preferences, some ingredients, food restrictions, nutrition, cost, etc. The server then returns this data and the client displays the filtered food items on the client, which the user can view.


User
Client
Server
Database
User clicks on filtering menu items option
Requests filtered items
Request menu items
Return menu items
Request relevant food item factors
Return relevant food items
Sort filtered items
Return filtered items
Display filtered menu items by specified factor

4.3.5 User searches for a food item

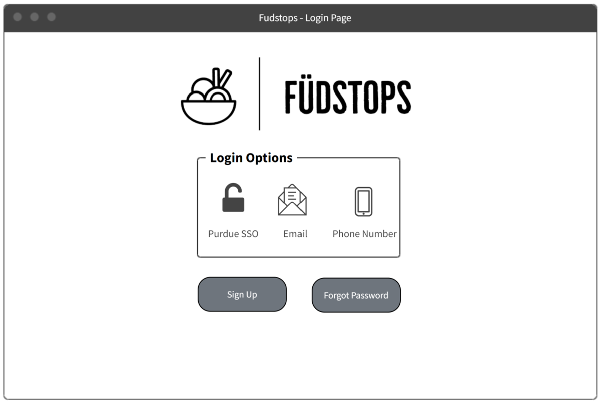
User searches for a food item, then the client requests the search query from the server. The server requests menu items from the database, which returns them to the server. The server then calculates a short list of possible matches to the food item requested, then returns the list of possible matches to the client. The client then displays the list of matches to the user.


User
Client
Server
Database
User searches for a specific menu item
Requests search query
Request menu items
Return menu items
Calculate list of possible matches
Return list of possible matches
Display list of matches to user

4.4 UI Mockups

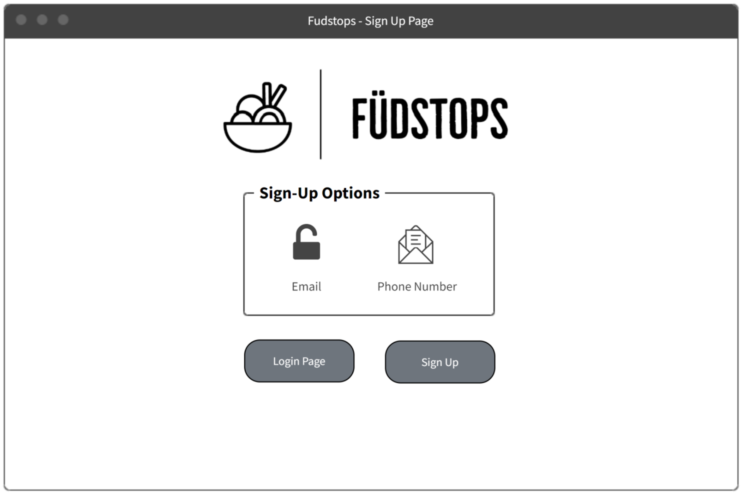
4.4.1 Login Page

After selecting the login method (sso/email/number), the middle box will alter to provide input fields for users to fill in and the buttons would alter to say “Login”. Then, the user would press the login button.



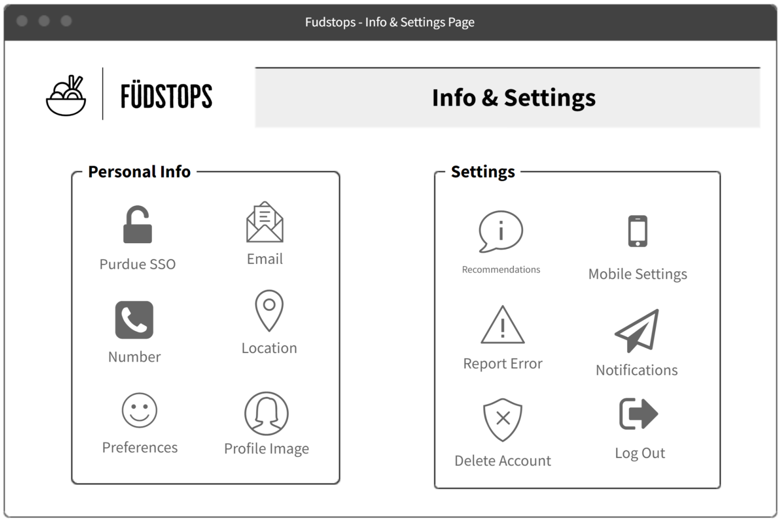
4.4.2 Sign-Up Page

After selecting the sign-up method (email/number), the middle box will alter to provide input fields for users to fill in. Then, the user would press the sign-up button.



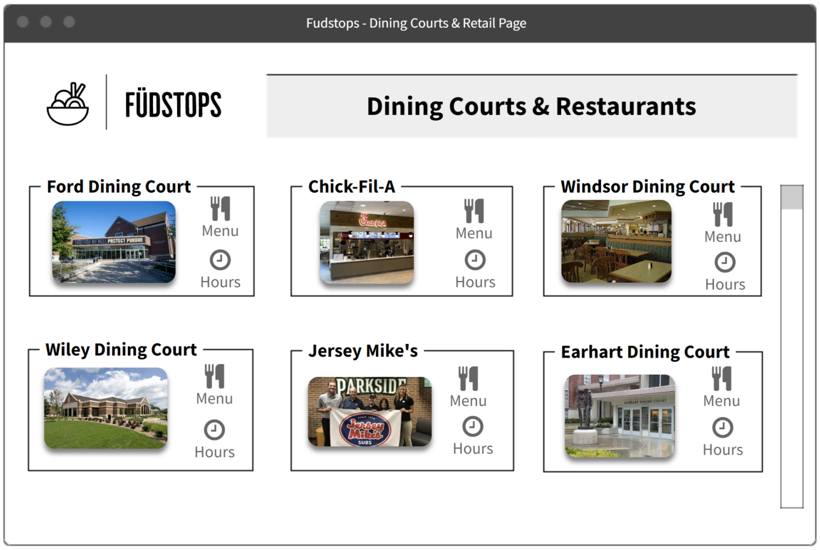
4.4.3 Page for personal information and settings (and to logout)

There will be various buttons for personal info & settings, and after a user clicks a button the user will be take to another page and can alter/view info there



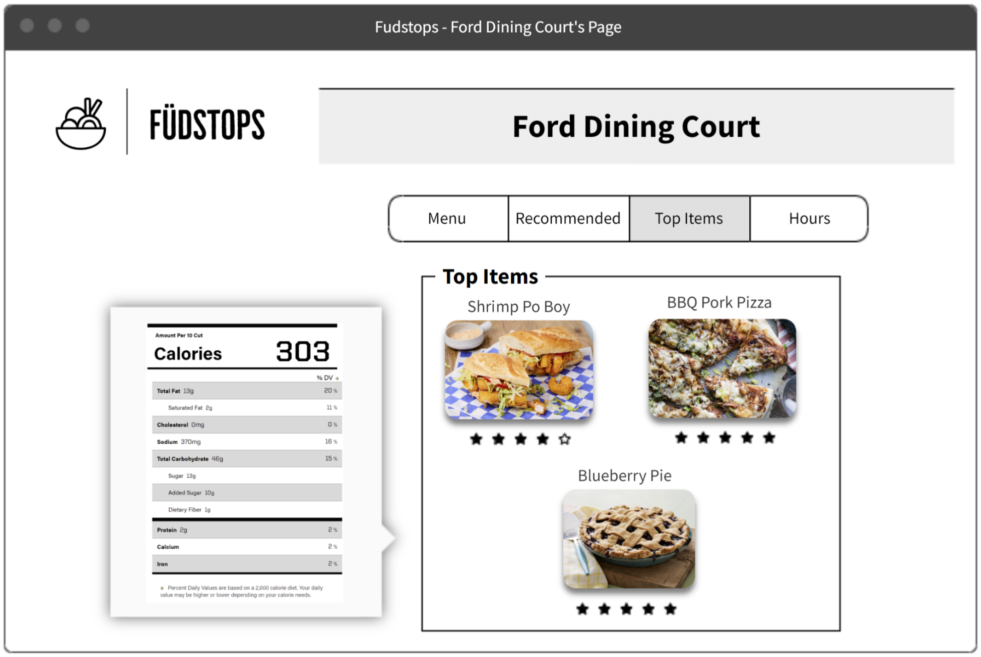
4.4.4 Page to display all the dining courts

Users can scroll (scrollbar on right side of page) through Purdue Dining Courts & retail locations and select the menu/hours buttons to be taken to the respective locations' information page.



4.4.5 One page that will show the menu & hours of a dining court

- There is a NavBar for navigation between Menu, Recommended Items, Top Items, and Hours for users to click through
- There is also a tooltip/popup to display the relevant info for each part of the NavBar
  - For items, dietary/nutrition info will pop-up
  - For hours, info about busyness will pop-up



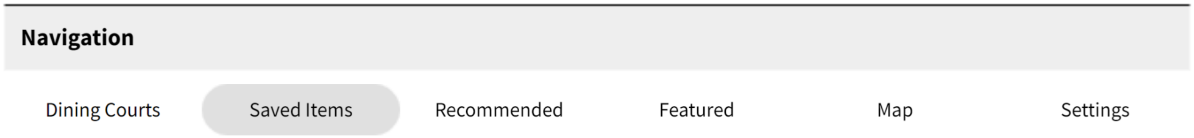
Amount Per 10 Cut

Calories	303
	% DV *
Total Fat 13g	20 %
Saturated Fat 2g	11 %
Cholesterol 0mg	0 %
Sodium 370mg	16 %
Total Carbohydrate 46g	15 %
Sugar 13g	
Added Sugar 10g	
Dietary Fiber 1g	
Protein 2g	2 %
Calcium	2 %
Iron	2 %

* Percent Daily Values are based on a 2,000 calorie diet. Your daily value may be higher or lower depending on your calorie needs.

4.4.6 Footer Component

There will be several tabs at the bottom to navigate through each category of page:


Navigation
Dining Courts
Saved Items
Recommended
Featured
Map
Settings