Team 11
Purdue Eats

Team Members:

Aniket Agnihotri

Anisha Sinha

Eric Thompson

Mark Jin

Sean Joo

Vaastav Arora

Team 11 Design Document

Table of Contents

Purpose

With the onset of COVID-19, Boilermakers have and continue to experience long wait times at Purdue's dining facilities and struggle to make good dietary decisions. Purdue residents often have trouble deciding on the best place to eat and spend their meal swipes at. Existing platforms solely provide solutions to one facet of the meal planning problem, either in the form of viewing available meals or tracking meal consumption, but not both. Additionally, these services lack integration with individual schedules and fail to incorporate an understanding of user preferences in selecting and consuming meals.

PurdueEats aims to utilize machine learning (ML) model(s) to understand a student's eating habits and provide them with a personalized dining experience at Purdue's dining facilities. PurdueEats is tailored towards user-specific dietary needs and provides more relevant and timely information to users compared to existing applications. Through PurdueEats, we hope to combine the many facets of meal planning and to add an element of personalization previously unseen, serving it all through a unified, intuitive interface presented as an easy-to-use mobile application.

Functional Requirements

1. Users can create, manage and delete their accounts. <u>As a user</u>
    a. I would like to be able to register for a PurdueEats account so that all of my information is associated with my account.
    b. I would like to be able to login to my PurdueEats account so that I can track my meals and view wait times.
    c. I would be able to login to my PurdueEats account via two-factor authentication so that my account is more secure.
    d. I would like to be able to change my account name or password so that I have control over my credentials.
    e. I would like my password to be reset when required so that I can regain access to my account if I forget my password.
    f. I would like to be able to change my registered email id so that I can have flexibility with my account credentials.
    g. I would like to be able to delete my user account so that my user-generated data is removed from the application's database.
2. Users can customize their profile and track the availability of their favorite meal. <u>As a user,</u>
    a. I would like to have a profile page so that I can edit my credentials, name, and meal plan as well as access my past meal history.
    b. I would like to upload a profile picture to my profile so that others can see me when finding a lunch buddy through the app.
    c. I would like to be able to mark meals as favorites so that I do not miss out on them when they are offered.
    d. I would like to be notified when my favorited meals are available and being served at a particular location so that I can track my preferred meals.

    e. I would like to be able to toggle notifications for each of my favorited meals so that I can customize the meals that I am tracking.
3. Users can easily view and access dining court information. As a user,
    a. I would like to be able to easily view meal options across Purdue's various dining facilities so that I may choose where to eat from the options I'm given.
    b. I would like to use a search bar to search through the dining court menus on any given day.
    c. I would like to be able to filter meals based on if they are vegetarian or contain specific allergens so that I can be sure that what I am eating fits my dietary needs.
    d. I would like to be able to view the wait times at each of Purdue's dining facilities so that I can decide where it would be quickest for me to eat.
    e. I would like to be able to view the distance from my location to each of Purdue's dining facilities so that I may account for distance to each facility as a factor in my dining decision-making.
    f. I would like to able to view a map of my current location and each dining court so that I can navigate to any dining court of choice.
4. Users can get personalized meal recommendations based on their requirements. As a user,
    a. I would like to be given meal recommendations that are consistent with my past eating habits so that I can stick to eating foods that I enjoy and have eaten in the past.
    b. I would like to be able to rate meals so that I get better meal recommendations based on my preferences.
    c. I would like to be able to input times that I am free to eat so that I am only recommended meals during those times.
5. Users can manage and view their past eating histories, including summaries. As a user,
    a. I would like to be able to view my past eating history so that I can keep track of my diet.
    b. I would like to be able to view the number of meal swipes that I have remaining for the week so that I can manage my meal swipe count better.
    c. As a user, I would like to be able to view my dining dollars balance so that I can track my transactions.
    d. As a user, I would like to be able to update and edit my dining dollars balance so that I can make necessary changes if my meal plan changes.
    e. I would like to view cumulative weekly summaries of what all users are eating through the app so that I can find popular eating trends.
    f. I would like to be able to see a wrap-up of the food that I have eaten over a semester so that I can look back on my eating habits for the semester.
    g. I would like to be able to view a visual representation of nutritional information regarding my meals so that I can have a clear understanding of the health impacts of the food I am eating.
6. Users can also look for lunch buddies. As a user,

   a. I would like to be able to match with other users to dine at locations that meet our mutual dining preferences so that I can meet new people while still abiding by COVID-19 regulations.
   b. I would like to be able to share my current location with my matched user so that I can meet up with them and dine together.
   c. I would like to have an "I'm here" button so that I can let my buddy know that I have arrived at the agreed location.
7. Users can review dining courts. As a user,
   a. I would like to be able to read reviews about dining courts so that I can learn more about a dining court from my fellow students before deciding on where to eat.
   b. I would like to be able to write reviews about dining courts so that I can share my experiences with others.
   c. I would like to be able to upvote or downvote reviews from others so that I can promote relevant reviews and demote irrelevant ones.
   d. I would like to be able to report reviews that are spam or inappropriate so that they can be taken down.
8. Accessibility, Usability, and Convenience features. As a user,
   a. I would like to have my login persist on exiting and re-entering the application so that I do not need to spend time logging in every time I would like to open the app.
   b. I would like to have a navigation bar so that I can access all of the features of the application from this screen.
   c. I would like to be able to adjust the app's text size so that I can clearly read information from the app.
   d. I would like to be able to view answers to frequently asked questions so that if I run into issues with the app I can find potential solutions.
   e. I would like to be able to send feedback to the creators of the application within the application so that if there are any issues the developers can fix them.
   f. I would like to be able to access a settings page so that I can manage my notifications, light/dark mode UI, text size, and fun fact feature.
9. Users can view daily fun facts. As a user,
   a. I would like to have a daily Purdue fun fact displayed each day that I log in so that I have an enjoyable user experience.
   b. I would like to disable the display of a Purdue fun fact so that I have a more minimalistic app experience.

Non-Functional Requirements

1. **Client Requirements:** As a developer,
   a. I want the frontend to be cross-platform and available on both Android and IOS devices.
2. **Server Requirements:** As a developer,
   a. I want the server to record past eating habits of users and log them into the database so that I can feed them to the GNN module.

   b. I want the server to fetch and save dining court and associated menus information to a database so that I can serve them on demand when required by the client.
   c. I would like to receive and record user-inputted wait time logs into the database so that I can use the data as a factor in our wait time prediction modeling dataset.
   d. I would like the API interface to be highly decoupled and broken down into independent routes, so that we can work on the routes parallelly.
3. **Design Requirements:** <u>As a developer,</u>
   a. I want the client and server to share a class structure that allows for seamless data transfer between them with the use of an API.
   b. I want the server, GNN, and XGBoost module to be deployed on independent VM instances so they can run asynchronously.
   c. I want the server, GNN, and XGBoost module to be containerized so that they are encapsulated, portable, and allow for more control of the permissions and resources used by each container.
4. **Performance Requirements:** <u>As a developer,</u>
   a. I want the application to support up to 40,000 concurrent users.
   b. I want to support up to 5,000 server request per minute with a maximum response size of 100 MB per request.
   c. I want the database to scale automatically to an upper bound of 1 TB of total disk data consumption.
   d. I want the prediction service to take <= 4s per query prediction for a dataset size up to 200 GB in size.
5. **Visual Requirements:** <u>As a developer,</u>
   a. I want the application to be visually appealing and easily navigable.
   b. I want a light/dark UI mode to reduce stress on the user's eyes.
6. **Security Requirements:** <u>As a developer,</u>
   a. I want the data collected by our app, specifically user ids, usernames, and passwords, to be secure both locally and in our database.
   b. I want the database to use a model of least privilege to further guarantee the security and integrity of stored user data.

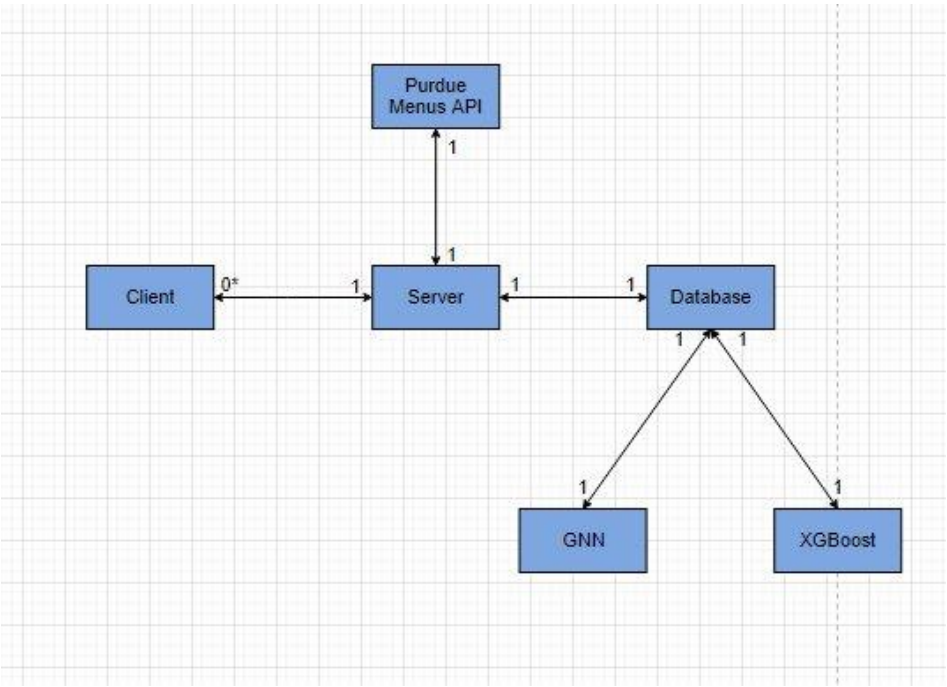
Design Outline

Our system consists of a cross-platform application that provides meal recommendations and meal planning services. It will be implemented according to the client-server architecture and uses a cloud database for persistent storage. Additionally, we will also use the service-oriented architecture style to support two machine learning modules that will run asynchronously on the server.

1. Client
    a. The PurdueEats client, running on both Android and iOS, will be distributed to users as an interface to our system.
    b. The client will send and receive data following the JSON file format via HTTPS requests to our RESTful API.
    c. Information sent and received by the client includes dining facility hours, meals, reviews, ratings, and user-specific information.
2. RESTful API
    a. The backend of PurdueEats will be created following the RESTful API architecture for ease of data transfer via HTTPS requests.
    b. The API will fetch data from both the PurdueEats database and the Purdue Menus API.
    c. The API will serve standard error codes in case of failures in order to conform to RESTful design.
    d. The API will additionally combine responses from the PurdueEats database and the Purdue Menus API, providing a singular, nested JSON response object to the client.
3. Database
    a. The PurdueEats database will store all user-specific data, reviews, and ratings.
    b. The database utilizes distributed cloud storage to ensure fast and consistent performance across various regions.
    c. The database will also scale memory and disk usage automatically based on user data volume over time.
    d. The database maintains one dynamic schema that grows with the number of menu items, in turn supporting the XGBoost module.
4. GNN Model
    a. PurdueEats' backend additionally hosts a Graphical Neural Network (GNN) model that consists of a bipartite graph with the set of users on one side and a set of menu items on the other.
    b. The two sets of nodes are connected by weighted edges with weights representing a certain user's rating for a certain menu item, with a value between 0-5.
    c. The GNN model uses this graphical embedding of eating history and ratings to recommend future meals based on user's preferences, the nutritional value of menu items, and their similarity with other users in the embedding.

5. XGBoost Model
    a. XGBoost is an implementation of gradient-boosted decision trees designed for speed and performance.
    b. This model can incorporate multiple mixed factors to predict numerical values. In our case, the numerical values to be predicted are wait times for each dining court. The factors in our case will include location, time of day, date, menu items, etc.
    c. Our model will generate a plot of predicted wait times based on such factors. We will overlay the plot with the actual wait time data to determine the accuracy of the prediction.

High-level Data Flow Model



Our system uses a one-to-many server-client architecture, allowing up to 40,000 users concurrently. When the client makes a request or sends data, it first gets sent to our backend server. This server either receives information from the Purdue Menus API, if it is menu information being received, or instead queries our database, which contains all user data. The GNN module uses database information from multiple users to generate recommendations that are then sent back to the database. The XGBoost module takes wait times from the database, uses them to calculate future wait times, and finally sends the calculation back. The GNN and XGBoost modules run concurrently, in real time.

Design Issues

Functional Issues:

1. Which navigation layout is best suited for the application?
    - **Option 1:** Hamburger Menu
    - **Option 2:** Tab Bar
    - **Option 3:** Scrollable Navigation

Decision: Option 2

Reasoning: A bottom tab bar proves to be the most sleek and effective way to navigate between application screens. Our application has five main pages: a home page (containing meal recommendations and menus), a map page, a find a buddy page, a profile page, and a settings page. A tab bar at the bottom of the screen works well, as the user has easy access to each of these pages that have their own icons, while the screen stays minimalistic and uncluttered. Since there are five different pages, scrollable navigation is counterintuitive, as the user will have difficulty keeping track of which page they are scrolling to next, making the user experience frustrating and time consuming. A hamburger menu has potential, but there are not enough main pages to navigate to that would warrant a button to display the various screens that could be navigated to. So, a tab bar provides a simplistic navigation layout that is effective for our use case.

2. How should users enter their past meals that they have eaten?
    - **Option 1:** Enter meals eaten in a text box
    - **Option 2:** Select meals via checkmarks for a particular dining court

Decision: Option 2

Reasoning: While there are fewer pages that the user needs to interact with if they can simply type the meals that they ate in a text box, they may not use the exact names of the meals causing issues in storing the data in the database and then using that data to make predictions for the future. The less variance there is in the data the more accurate recommendations will be for the user. It is also more convenient for the user to tap the meals that they ate as opposed to recalling and typing it out. Furthermore, option 2 offers a smoother UI and offers a more minimalist look.

3. How should users let their lunch buddy know that they are at the dining facility?
    - **Option 1:** Create a messaging service within the app so that users can communicate with agree on a location to meet.
    - **Option 2:** Have the two users share their physical location with each other.
    - **Option 3:** Have an “I’m here!” button that each user can press to send a notification to the other user to let them know that they are at the dining location.

Decision: Options 2 and 3

Reasoning: A messaging service allows for two users who have matched as lunch buddies to discuss where and how to meet and dine together; however, it opens up a host of problems, such as appropriateness of messages and regulating communication between two users. Hence, we have chosen to simplify the mode of communication by limiting it to sharing only a user's current location, which allows a user to track where their lunch buddies are. Furthermore, we combine this option with the option of having an "I'm here" button, which notifies a user's lunch buddy that they've arrived at the recommended dining location.

4. How should wait times be relayed to users?
    - **Option 1:** Show popular times similar to Google analytics
    - **Option 2:** Show the predicted wait time for the given time of day.

Decision: Option 2

Reasoning: The purpose of PurdueEats is to help users streamline their meal planning process. By providing the user the predicted wait time at the facility, they can make quick decisions during the moment and either get their meal now or later. Furthermore, it will be easy for the user to compare the wait times for each dining facility at any given time. Showcasing popular times will clutter the screen and force the user to analyze the various times during the day and what will likely work for them based on their schedule, as well as compare each of the dining courts' popular times to find the shortest queues.

5. Should dining dollars be pulled from the user's Purdue account periodically or should the user manually enter their transactions?
    - **Option 1:** Periodically request the number of dining dollars in the user's account.
    - **Option 2:** Have the user enter their initial dining dollars and then every transaction where they purchase an item.

Decision: Option 2

Reasoning: Having the dining dollars be updated periodically would be be less work for users, as they would only have to update their dining dollar info every so often. However, because the dining dollars are not updated constantly, the value in the account can be inaccurate. Users may also become annoyed getting a dining dollar update notification if they are in a hurry or do not wish to use the feature. While having the user enter their dining dollars every transaction is more work for the user, it guarantees that the dollar amount will be accurate if it is being updated. Also, users will not have to deal with notifications.

Non-Functional Issues:

1. How are we going to host and deploy our server and models?
    - **Option 1:** Google Cloud Platform
    - **Option 2:** Amazon Web Services

Decision: Option 1

Reasoning: Google Cloud Platform (GCP) and Amazon Web Services (AWS) are both top contenders on the cloud computing services page. Both services support cloud VM instances to train our models and relational cloud databases with distributed querying in the form of BigQuery and Amazon RDS, respectively; however, GCP has better cohesion with all its different services, integration with GitHub code repositories, Jupyter Notebook, and has better support for Tensorflow VM containers, making GCP the better choice.

2. What model are we going to use to predict the line length and wait time of each dining court?
    - **Option 1:** Time-Series, e.g. ARIMA model
    - **Option 2:** XGBoost
    - **Option 3:** Artificial Neural Networking (ANN)

Decision: Option 2

Reasoning: Time series may be the intuitive solution, but we are unable to obtain a good model that takes into account multiple factors, making generating predictions with high accuracy difficult. ARIMA works well if only one factor is used, but as soon as multiple factors come into play, the predictions from the model become rather unreliable. ANN can be a viable solution, but generally, NNs are difficult to understand, explain, and have longer run-times. We prefer NNs as a fall-back plan. XGBoost is the best better option, compared to ANN, since it is decision-tree based, incorporates multiple factors very well, has faster run-time, and produces surprisingly high accuracy most of the time.

3. What model are we going to use to analyze past eating habits and to make meal recommendations?
    - **Option 1:** Collaborative Filtering
    - **Option 2:** Deep Neural Network
    - **Option 3:** Graphical Neural Network

Decision: Option 3

Reasoning: Out of the three options, collaborative filtering is the simplest model and fastest to train, but it does not generalize well to new data (new users and new menu items added to the database), making it unfit for our use cases. Deep Neural Networking

on the other hand does generalize fairly well to new inputs, but it performs offline learning and cannot accommodate new data points (new user-menu item ratings) without having to completely retrain. Graphical Neural Networks solve this problem as they perform online learning, generalize well to new inputs, and don't require any additional effort to train compared to Deep Neural Networks, making them the best choice for our use case.

4. What frontend language should be used?

- **Option 1:** React-Native
- **Option 2:** Swift
- **Option 3:** Kotlin/Java

Decision: Option 1

Reasoning: Rather than build either an iOS (Swift) or Android (Kotlin) application, we chose to use React-Native as it is a multi-platform framework. Kotlin has recently introduced a multi-platform option, however it is still fairly new and experimental. React-Native has more established documentation as well as many libraries to use. In addition, it makes the development process seamless, regardless of the devices that each developer has, as the application can be simulated on both iOS and Android products.

5. What/how should data should be incorporated into the XGBoost model for wait time prediction?

- **Option 1:** Merge existing historical wait time dataset (columns include Location, Timestamp, Wait time) with historical Dining Facility Menu Items dataset (including Location, Date, and array of Menu Items) by Location and Timestamp/Date
- **Option 2:** Merge existing historical wait time dataset (columns include Location, Timestamp, Wait time) with historical Dining Facility Menu Items dataset (including Location, Date, and array of Menu Items) by Location and Timestamp/Date, with Menu Items encoded into individual columns per menu items
- **Option 3:** Merge existing historical wait time dataset (columns include Location, Timestamp, Wait time) with historical Dining Facility Menu Items dataset (including Location, Date, and an array of Menu Items) by Location and Timestamp/Date, with Menu Items first grouped into item group (e.g. "Chicken Masala", "Oven Roasted Chicken Breast", "Chicken Noodle Soup" categorized into group "Chicken"), with groups encoded into individual columns per menu items

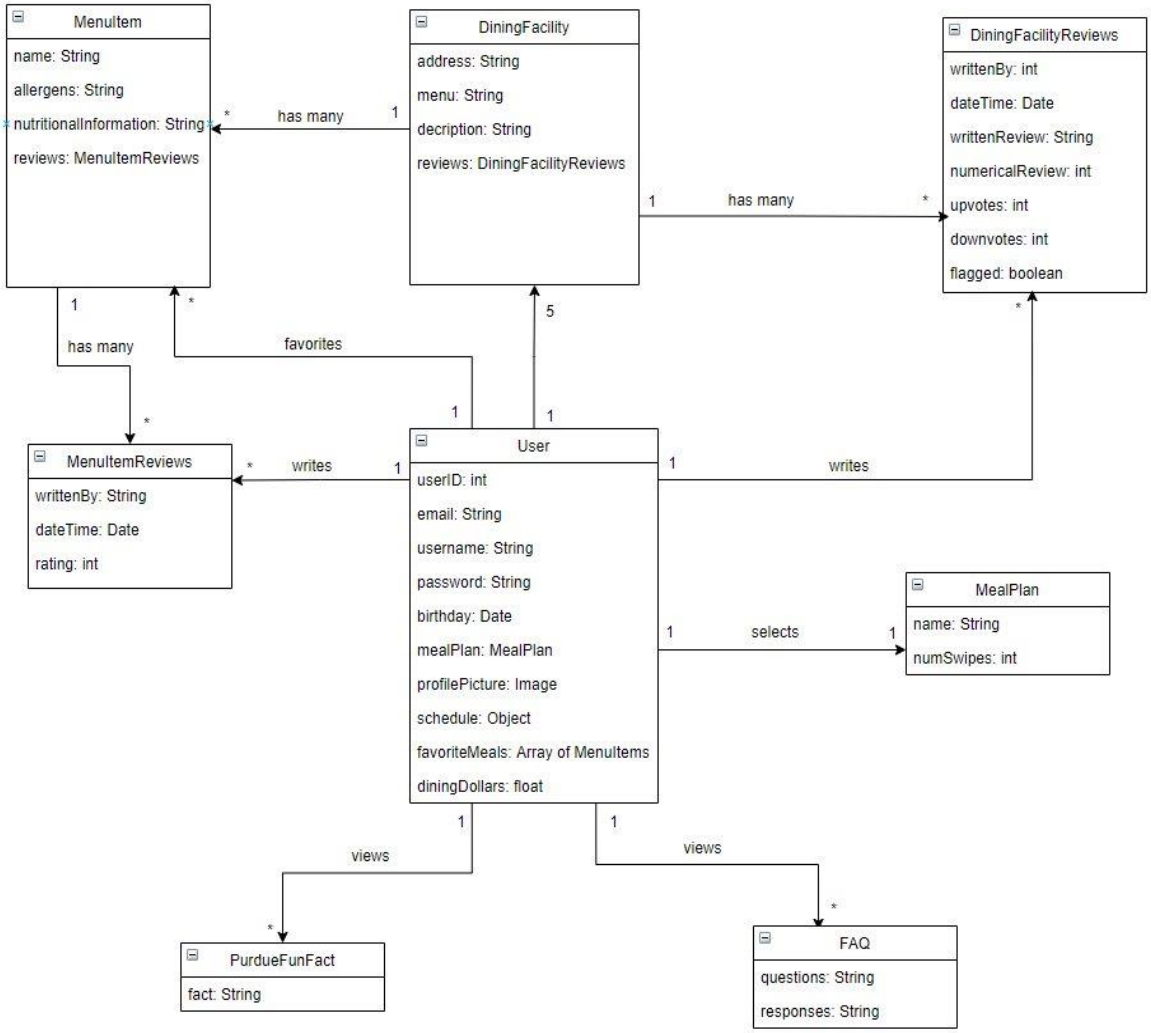
Decision: Option 2

Reasoning: XGBoost requires encoding of factors involved in the dataset, so option 1 is out of the question as the XGBoost library does not have functions that parse arrays into factors. With option 2, we will encode each dining menu item as a column (factor) per

item in the merged dataset. Values of each column will be encoded binary-style based on whether the dining facility contains the menu item. Since the number of factors will inversely affect the accuracy of the model, the more the factors, the less accurate we can expect the model to be. With option 2, we expect possibly up to 200 individual menu items. With our sample size of 3000 entries, the decrease in accuracy is a concern. If the accuracy of results under this option is too low, we will transition to option 3, which sacrifices the granularity of menu items for an increase in accuracy.

Design Details

Class Diagram



Description of Classes and their Interactions

Our Class Diagram is heavily based on the efficient and suitable database representation of the information gathered and served on our frontend. These classes will mainly serve as blueprints for the classes supported by our API upon receiving data and for the JSON file packages that are passed from our API to our frontend. Our Class Diagram additionally shows inter-class relationships and how they correlate according to frontend user interaction.

DiningFacility

- Represents all of the dining facilities at Purdue and each respective addresses, menus, and descriptions.

DiningFacilityReviews

- Represents the reviews for each dining facility.
- The reviews will be on a zero to five-star scale as well as a description justifying the user's rating. This description can be upvoted, downvoted, and flagged by other users whenever applicable.

FAQ

- Represents the frequently asked questions regarding Purdue dining facilities.

MealPlan

- Represents the four different meal plans offered at Purdue.
- Users will be able to select their meal plan so that whenever they make a transaction, the application can make necessary adjustments to the user's meal plan counter.

PurdueFunFact

- Represents a fun fact specifically about Purdue.
- This will be represented as a banner that can be turned off if preferred and will be refreshed daily.

MenuItem

- Represents the different foods on the menu.
- Each menu item will have specific allergens when applicable.
- Each menu item will have corresponding nutritional information.

MenuItemReviews

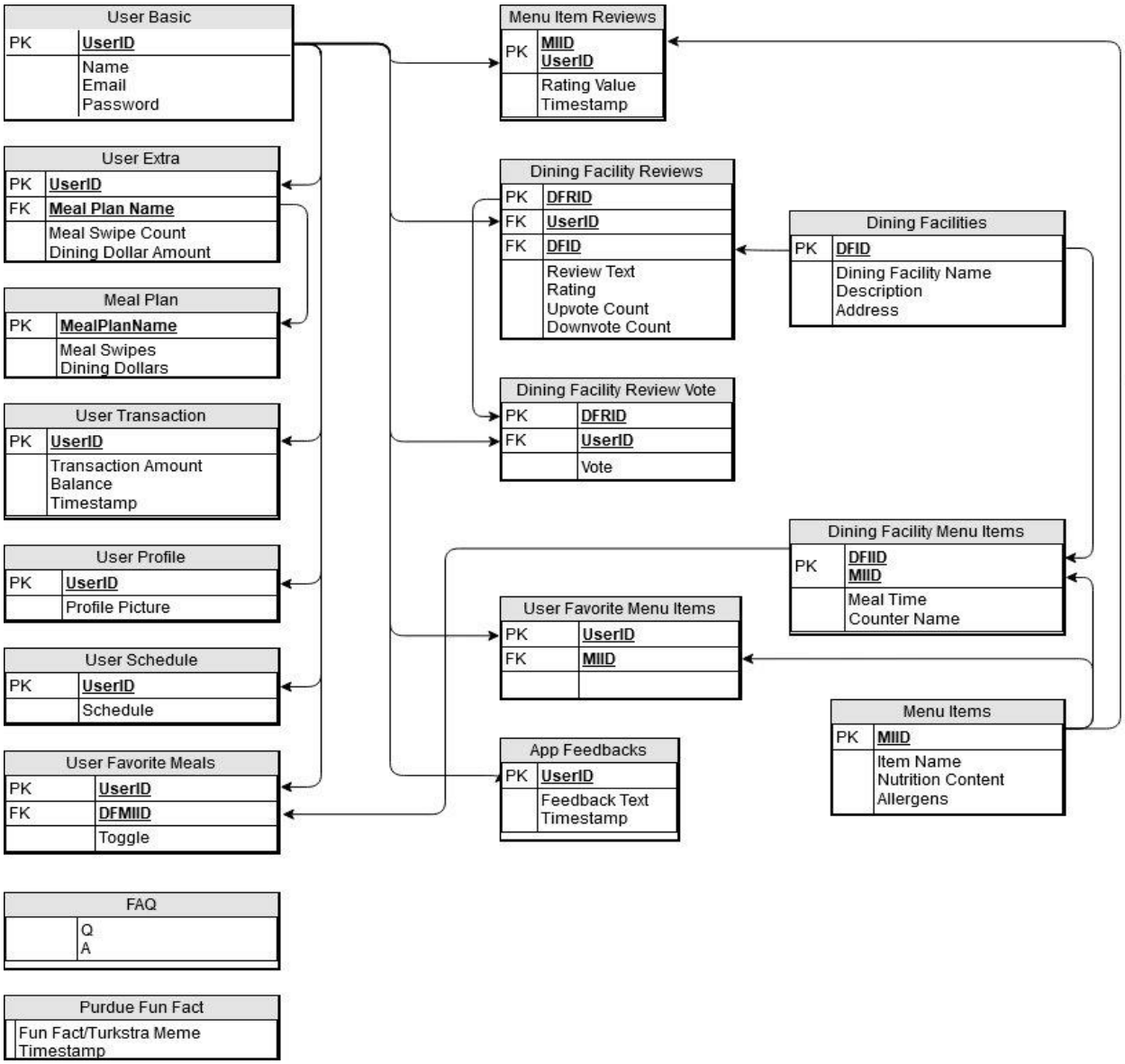
- Represents the reviews for the meal the user had
- When creating a review, the user's ID and Vote value will be taken in as arguments.
- The reviews will be on a zero to five-star format.

User

- Represents someone that has registered an account on our application.
- Each user has a specific ID.
- Each user will input their own username, password, and email so that they can login.
- Each user will have the option to input/update a profile image.
- Each user will have their schedules accessible.
- Each user will have access to their favorited meals
- Each user can create their feedback
- Each user's current meal plan will be visible.
- Each user's transactions will be taken into account and be subtracted appropriately.

Database Design

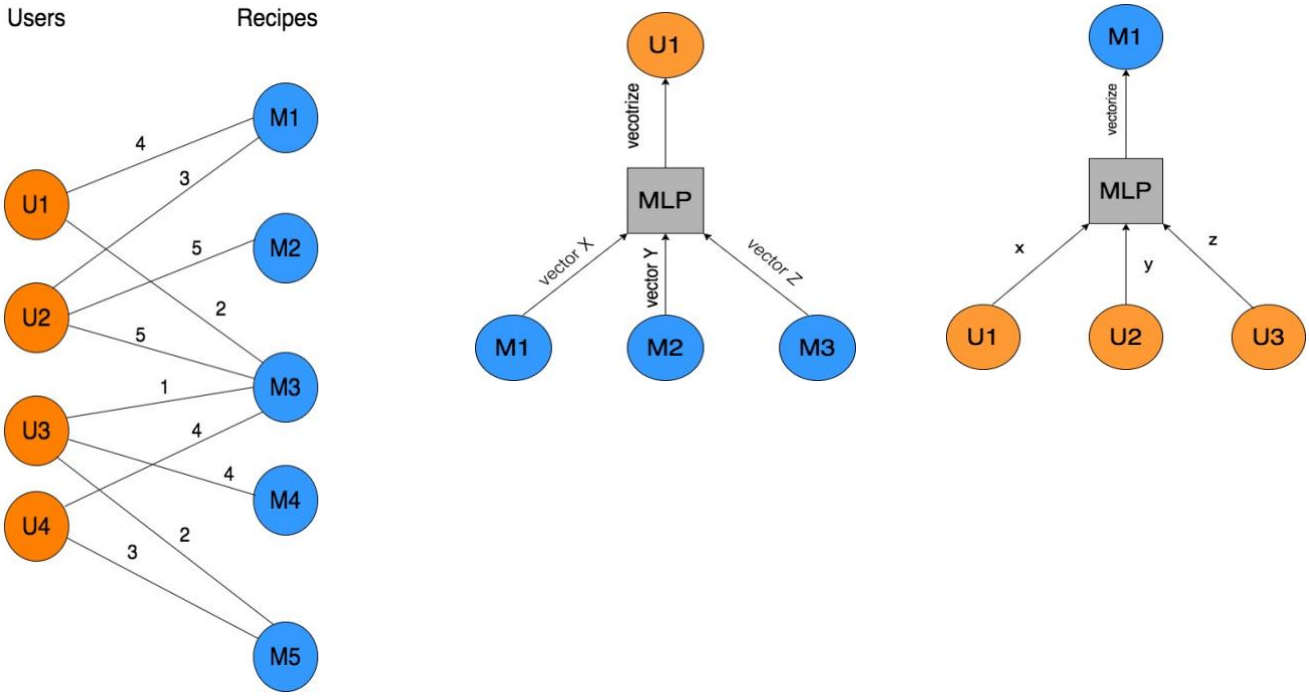
We are using a relational database design with MySQL as our database management system (DBMS) of choice and BigQuery as our cloud hosting solution. Our schema has a table design to mimic a near one-to-one mapping with our data class design, with a table in the database for each of the data classes that we identified in our domain. The rows of each table are indexed with a Primary Key. We use auto-generated unique IDs as Primary Keys in several of our relations that are produced on demand by the server. Some relations use more complex Primary Keys such as a combination of two ID attributes, based on the uniqueness constraints of that table. Relationships between the tables are created using these Primary Key -Foreign Key constraints and enforced implicitly by the DBMS.



Graph Neural Network

Graph neural networks (GNNs) are connectionist models that help embed relations through the means of graph nodes and edges; they capture the dependence of graphs via message passing between the nodes of graphs. Unlike standard neural networks, graph neural networks retain a state that can represent information from its neighborhood with arbitrary depth. For our application, we utilize a general inductive framework that leverages node feature information to efficiently generate node embeddings for previously unseen data, called GraphSAGE. We learn a function that generates embeddings by aggregating features from a node's local neighborhood and easily accommodates new nodes and edges in the future.

For our application, we embed users and menu items as nodes and the rating users assign to menu items as edges between the nodes. This leads to the construction of a bipartite graph with the set of users on one side and a set of menu items on the other as shown below. The **U** nodes represent users and **M** nodes represent menu items. The first pass of our algorithm analyzes the neighborhood of a menu item and creates a menu item vector for it. The second pass of our algorithm analyzes the neighborhood of a user and creates a user preference vector utilizing the previously generated menu item vectors. Finally, we run one last non-linear transformation on the required user vector and menu item vector to generate an expected rating prediction. This model will be implemented using Tensorflow 2.0, as it supports efficient matrix representation and computations, and integrates seamlessly with our python workflow.

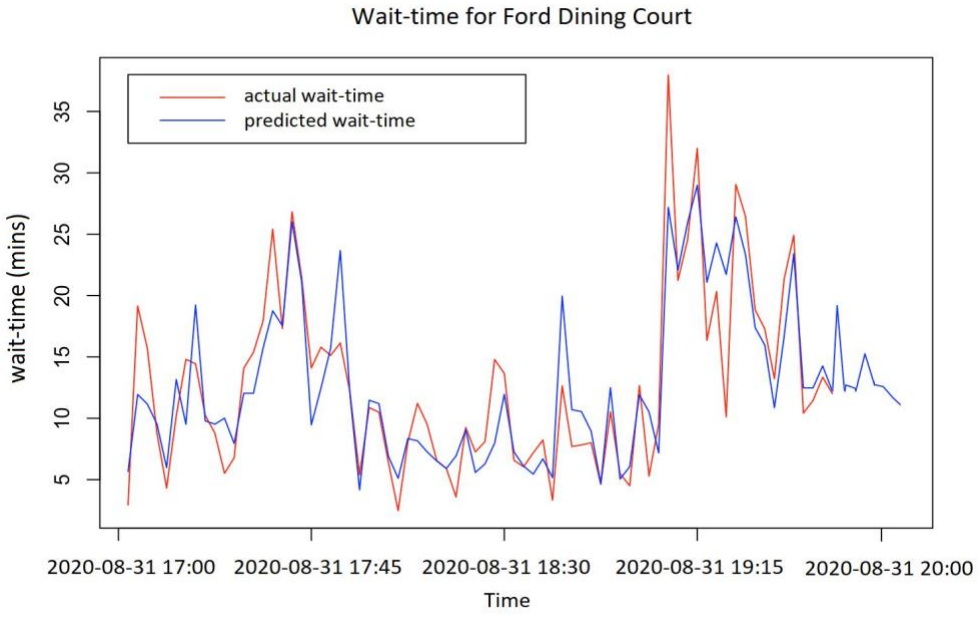


XGBoost Module

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM). Boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. Parallel tree boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. A popular example is the AdaBoost algorithm that weights data points that are hard to predict.

We are implementing XGBoost for its execution speed and model performance. Generally, XGBoost is superior in execution speed when compared to other implementations of gradient boosting. XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems. The evidence is that it is the go-to algorithm for competition winners on the Kaggle competitive data science platform.

For our application, we will utilize the historical dataset of user-submitted wait time estimations per dining facility and time of day, and merge it with the historical dataset of dining facility menu items. We will encode each dining menu item as a column (factor) per item in the merged dataset. The model will generate a predicted wait time for each dining facility. We will overlap the generated plot with the plot of the actual wait times based on user-submitted reports.



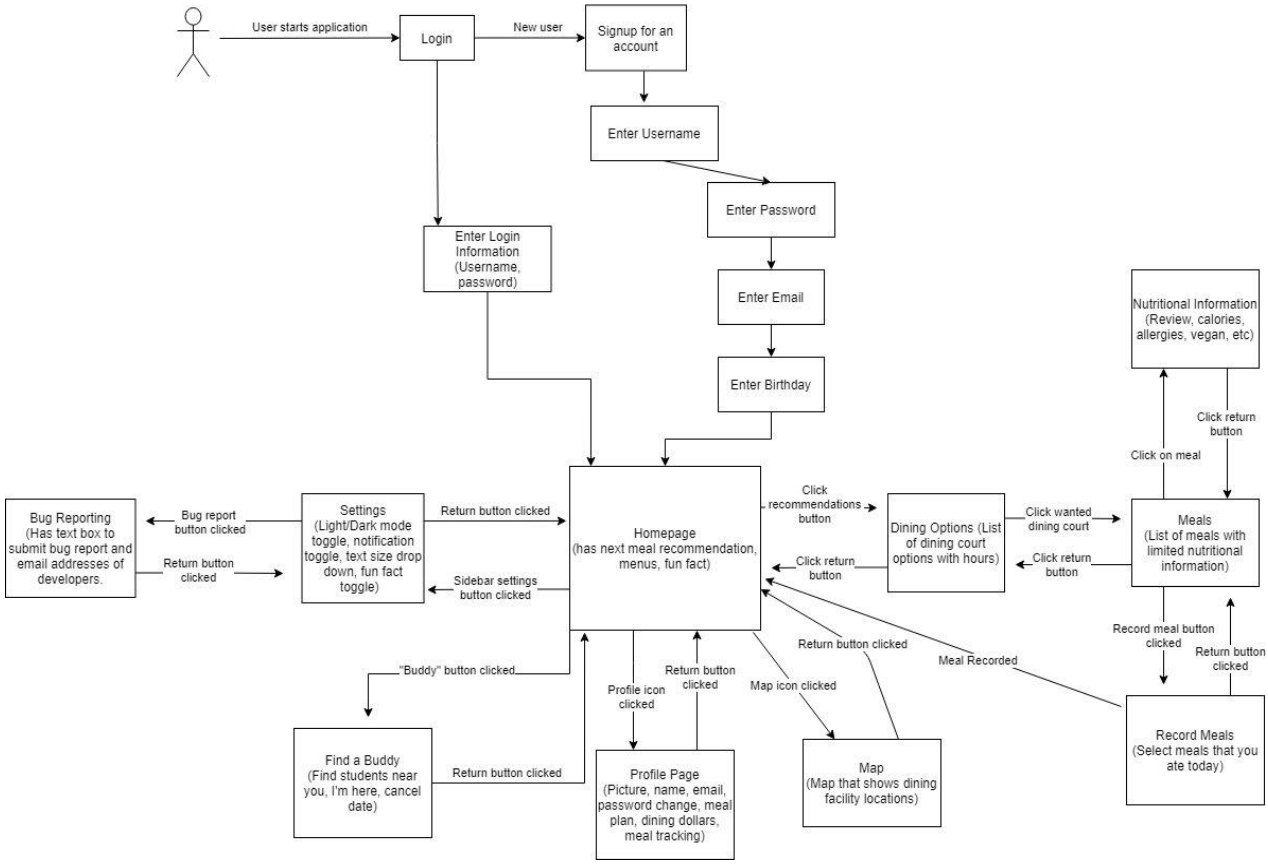
API Routes

The API routes are implemented so that the Uniform Resource Identifiers (URIs) can be utilized by the API server to ensure seamless communication between the frontend and backend. Moreover, there are different supported HTTPS methods for different resources that will be employed, such as DELETE, GET, POST, and UPDATE. When is a resource is read-only, it will support only GET requests, whenever a resource is readable and writable (creatable), it will support POST and/or DELETE requests, and if a resource is editable, it will support UPDATE requests. Finally, every resource is accessible/modifiable by its specified route as mentioned below on the API routes table.

Column 1	Column 2
Route	**Supported HTTPS methods**
/DF	GET
/DF/{DiningFacilitiesID}	GET
/DF/{DiningFacilitiesID}/Menu	GET
/DFR	GET
/DFR/{DiningFacilitiesID}/	GET, POST
/DFR/{DiningFacilitiesID}/{ReviewID}	DELETE, GET, UPDATE
/DFR/{DiningFacilitiesID}/{ReviewID]/Vote	DELETE, GET, UPDATE
/FAQ	GET
/MP	GET
/MP/{MealPlanName}	GET
/PFF	GET
/PFF/{Date}	GET
/MenuItems	GET, POST
/MenuItems/{MenuItemsID}	GET
/MenuItems/{MenuItemsID}/Allergen	GET
/MenuItemReviews	GET, POST
/Users	DELETE, GET, POST
/Users/{UserID}	GET, UPDATE
/Users/{UserID}/Auth	GET, POST
/Users/{UserID}/FavMeals	GET, POST, UPDATE
/Users/{UserID}/Feedback	POST
/Users/{UserID}/MealPlan	GET, POST
/Users/{UserID}/ProfilePic	GET, POST
/Users/{UserID}/Schedule	GET, POST
/Users/{UserID}/Trans	GET, POST

Navigational Flow Map

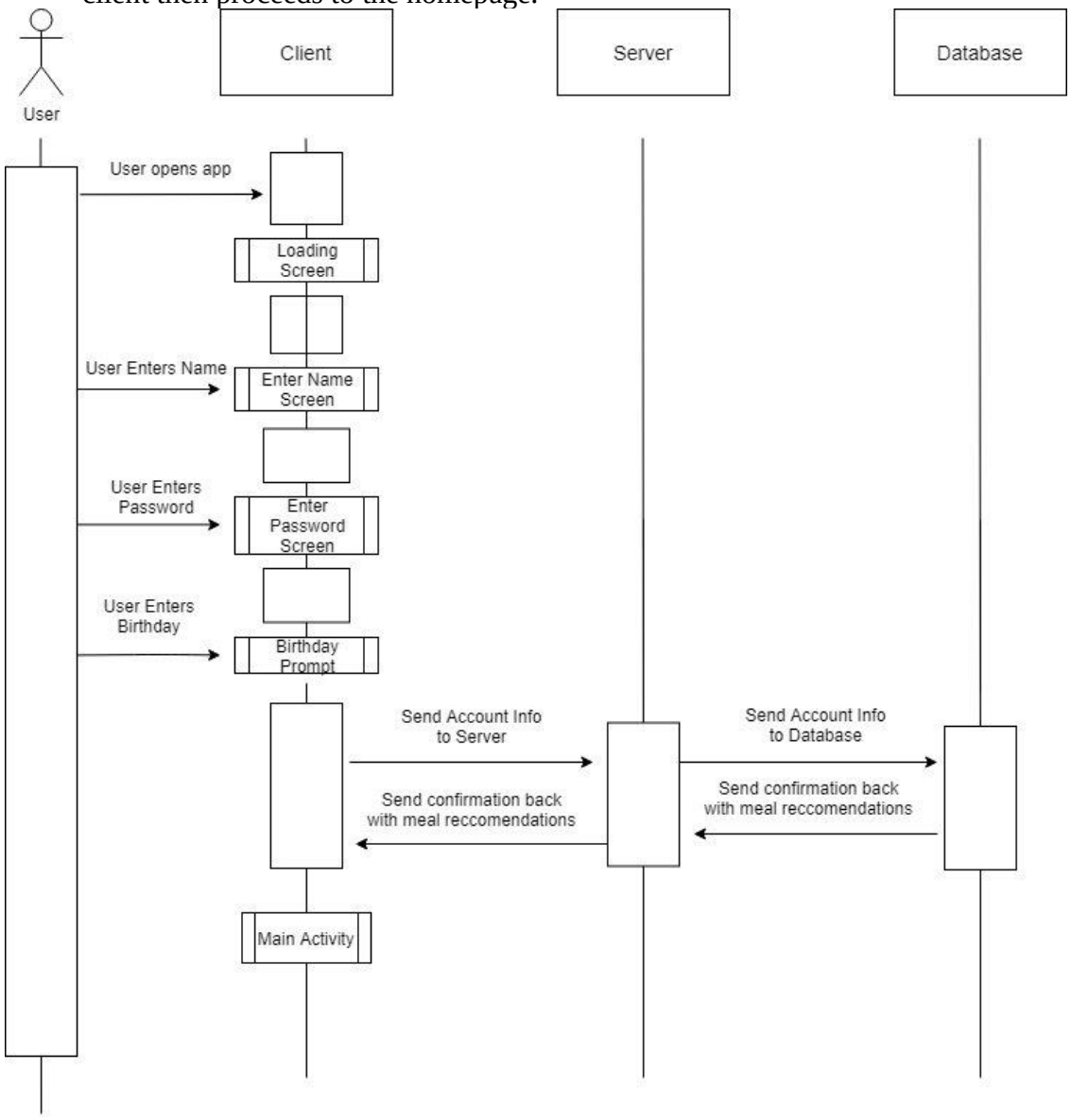
A flow map representing the screens and screen transitions of our application.



Sequence Diagrams

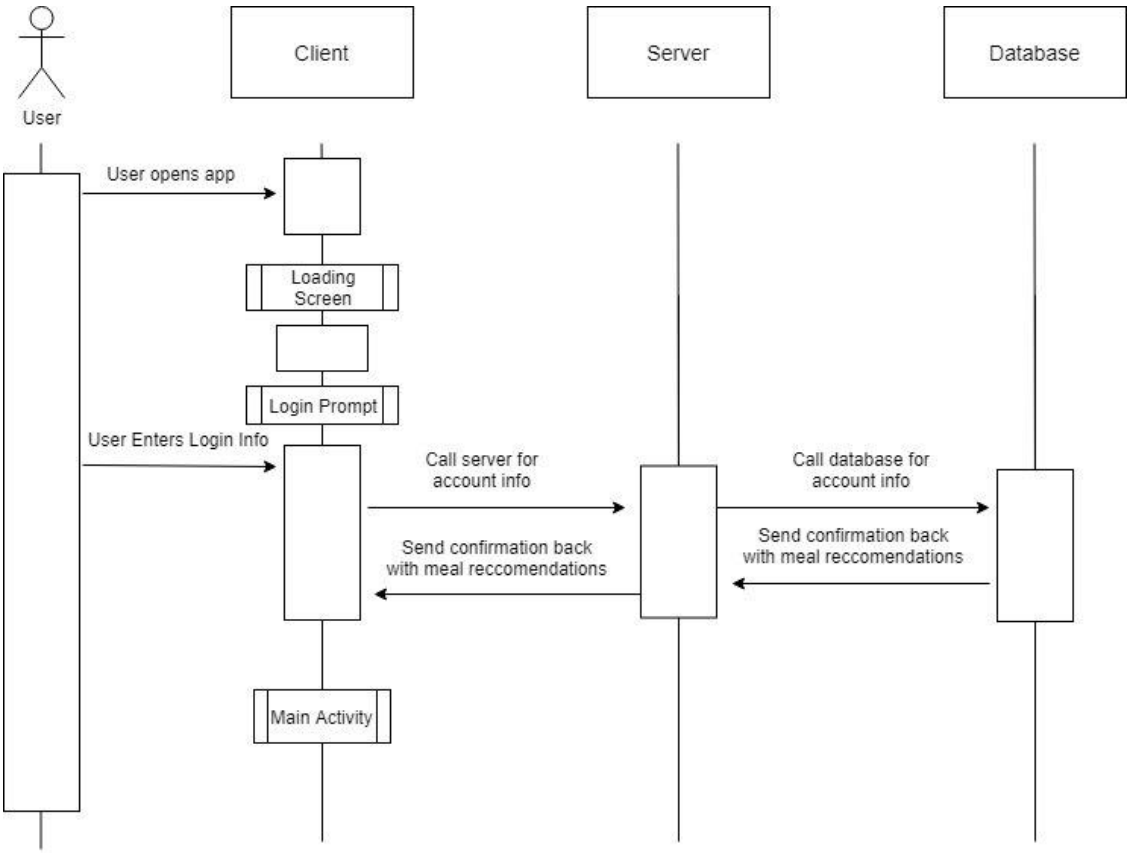
First Time User Registration

- The user opens the app and is prompted for account info on a series of screen. This information is uploaded to the database and a confirmation is sent back. The client then proceeds to the homepage.



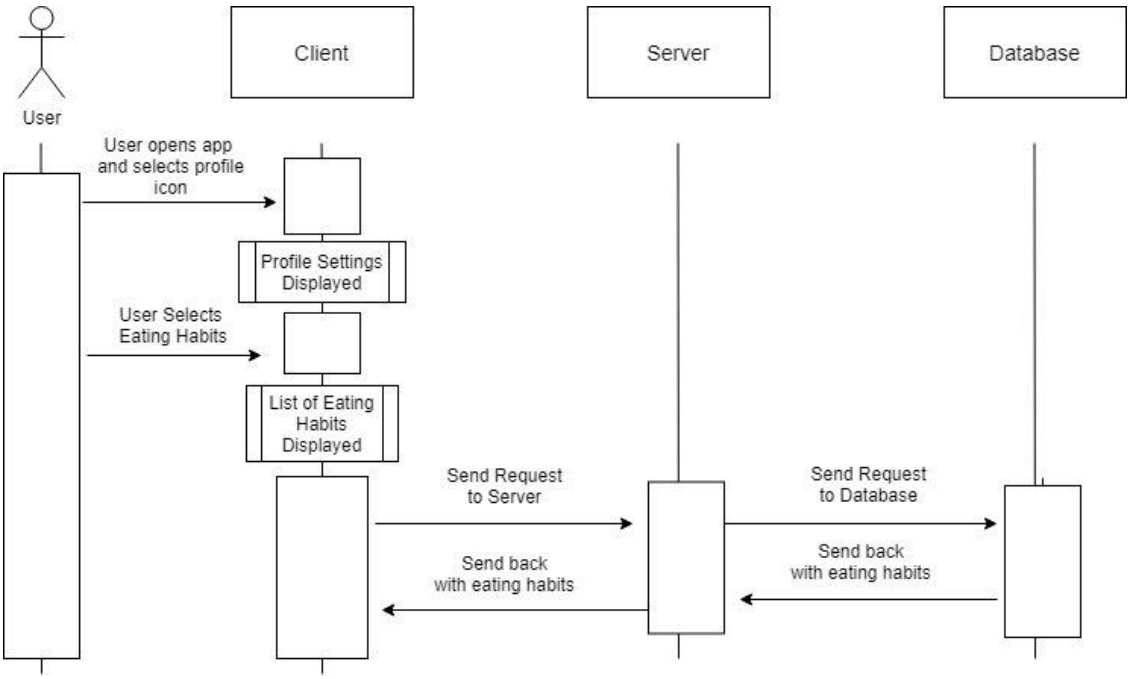
User Login

- The user is prompted for their login information. After the user enters their information, the account details are checked against the database. If the login is valid, the user is taken to the homepage.



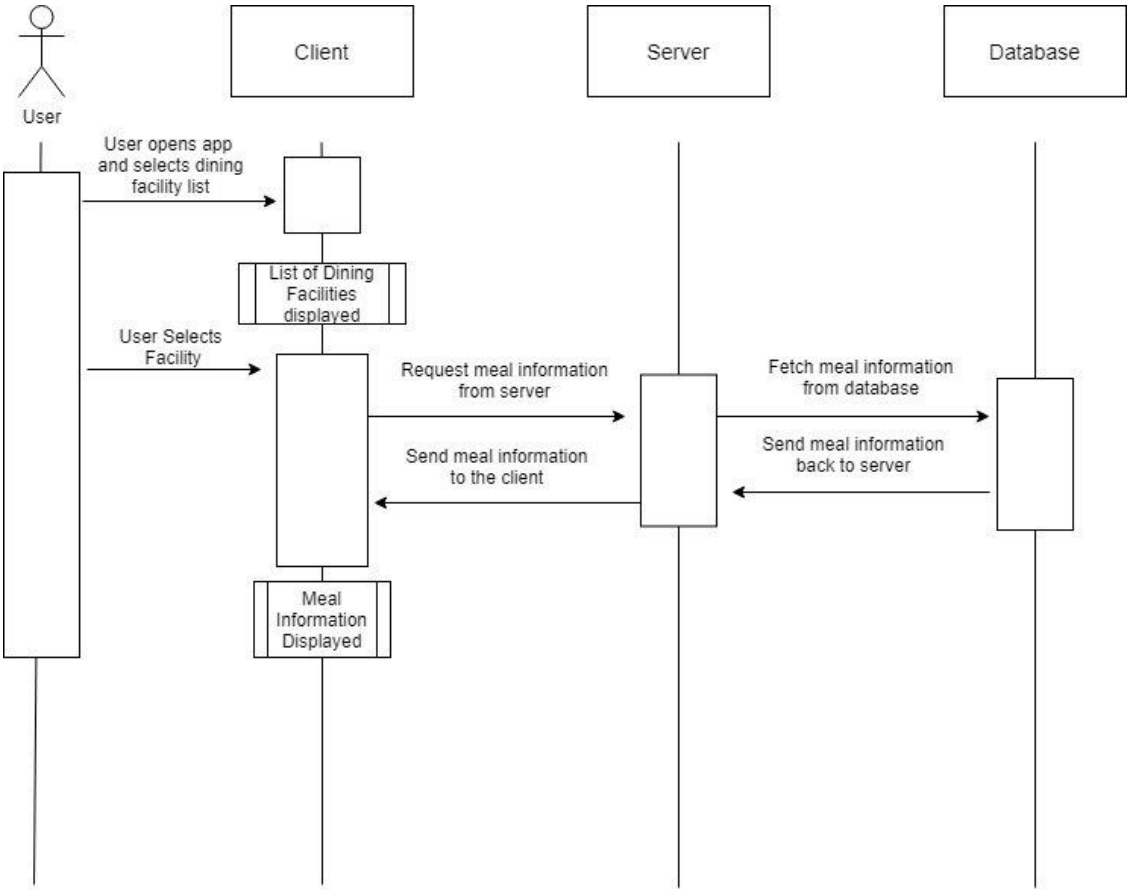
User Views Meal Recommendation

- The user clicks the profile icon and is taken to the profile page. They then select eating habits, prompting the app to send a request for the information from the database. The information in the database is sent back to the application for the user to view.



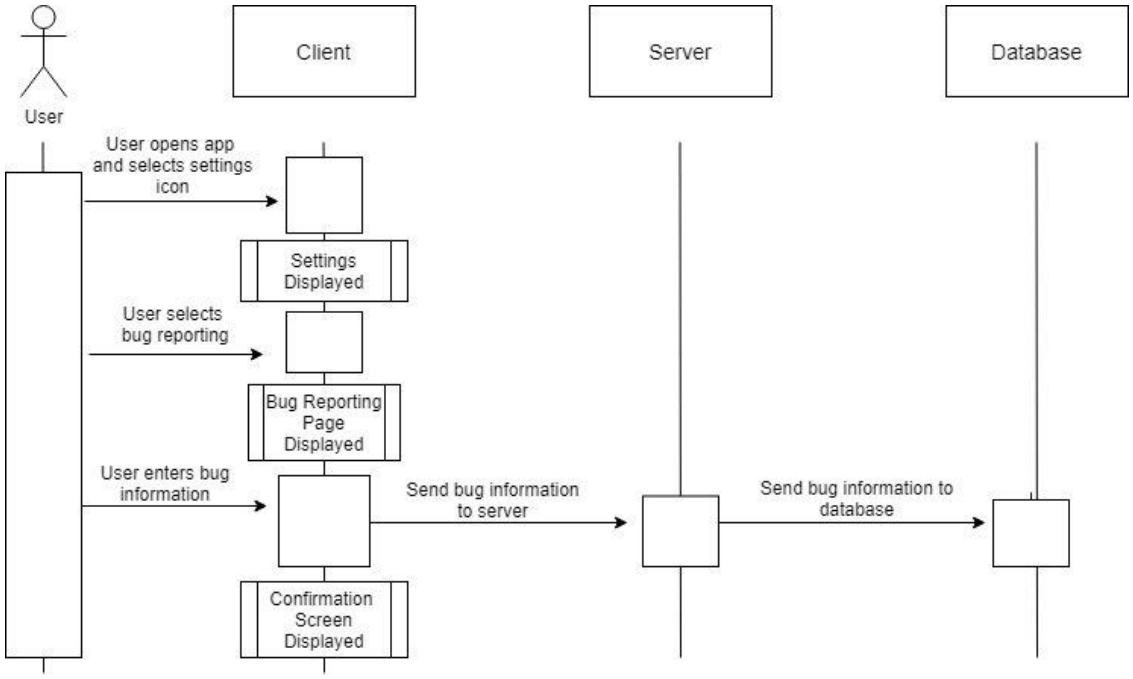
User Navigates to Meal List

- The user clicks the meals tab on the homepage to bring up the list of dining facilities. The user then selects the icon for the facility they are interested in and the meal information is displayed on the client.

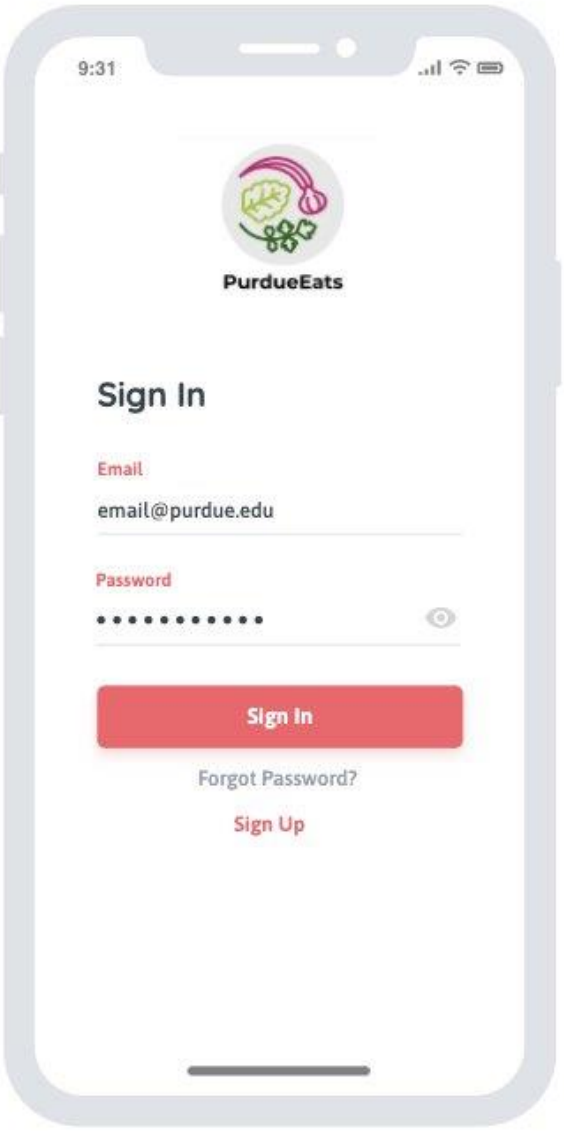


User Navigates to Bug Reporting

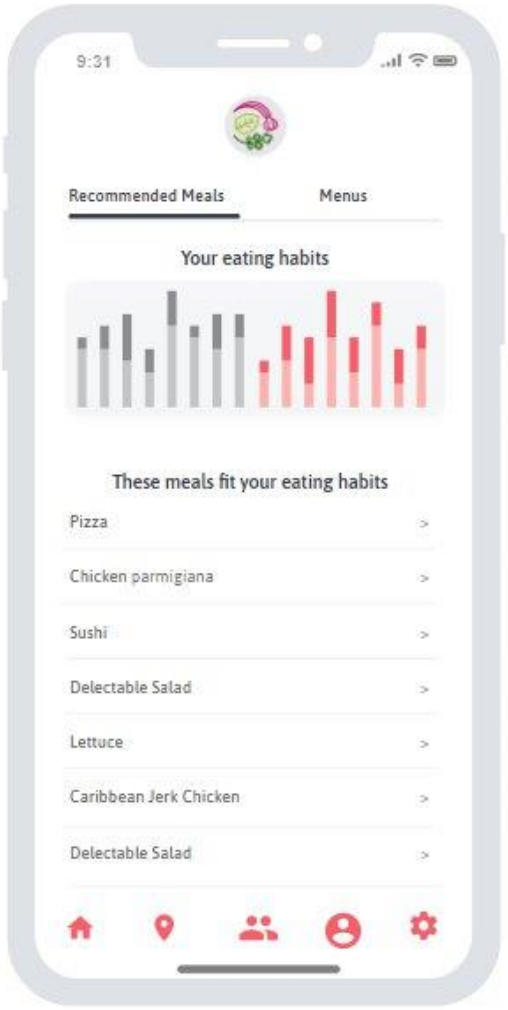
- The user redirects to the settings page and clicks the “Report a Bug” button. The user is directed to new screen to type out their bug report and press send.



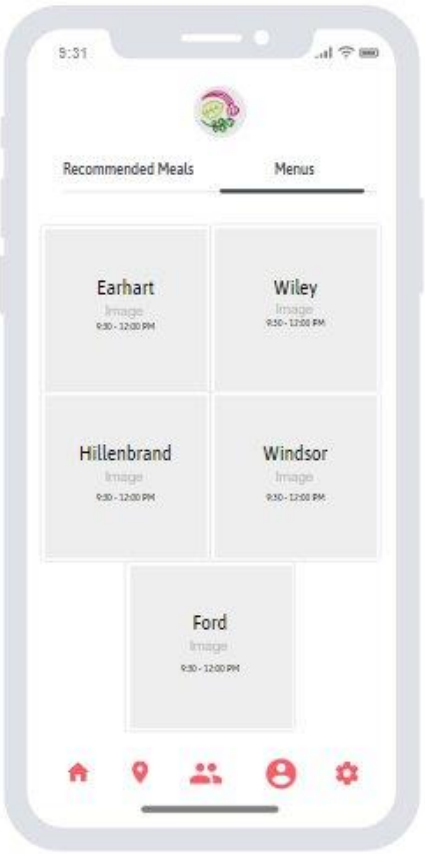
UI Mockups



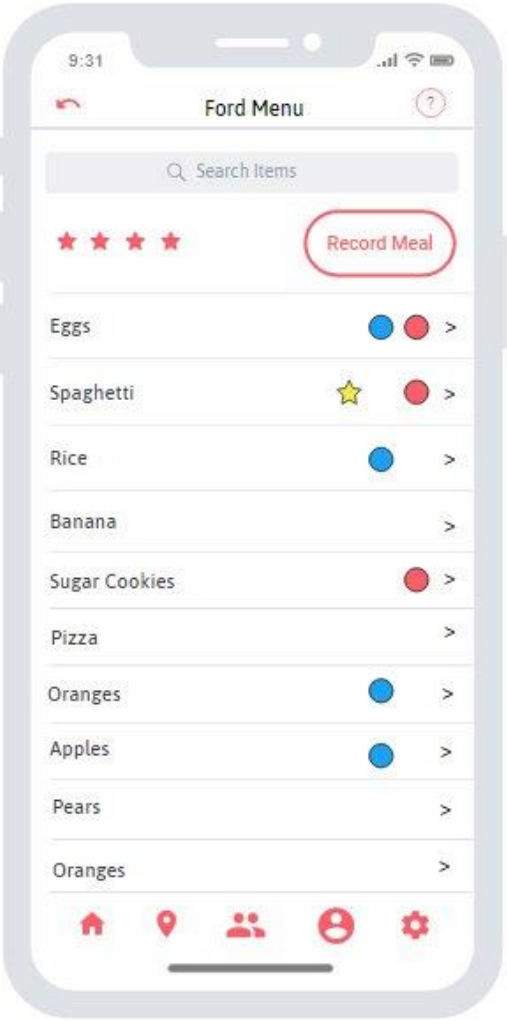
The sign-in screen will allow the user to sign in with their email and password or sign up for an account.



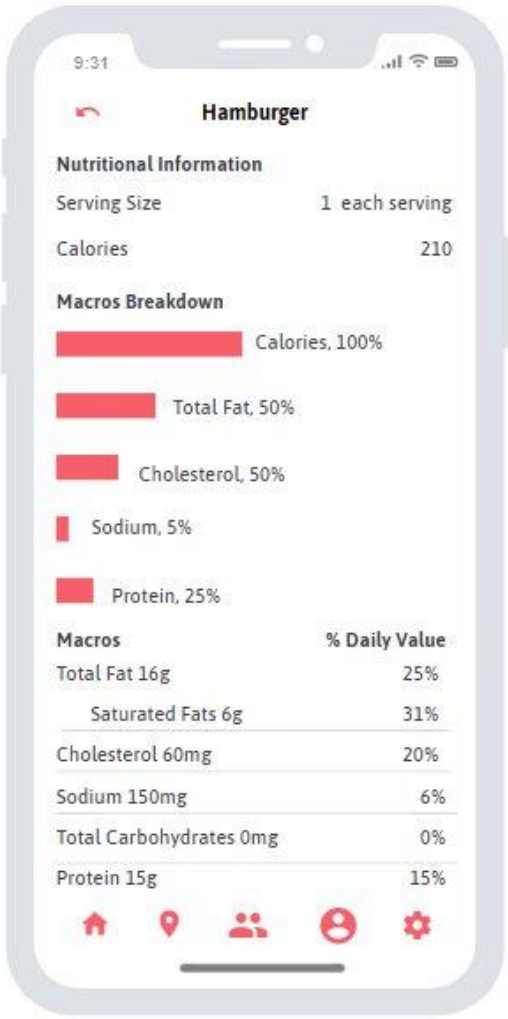
The user's personalized Recommended Meals tab provides a chart that will showcase frequency of past eating habits. A personalized list of meal item recommendations are displayed for the user.



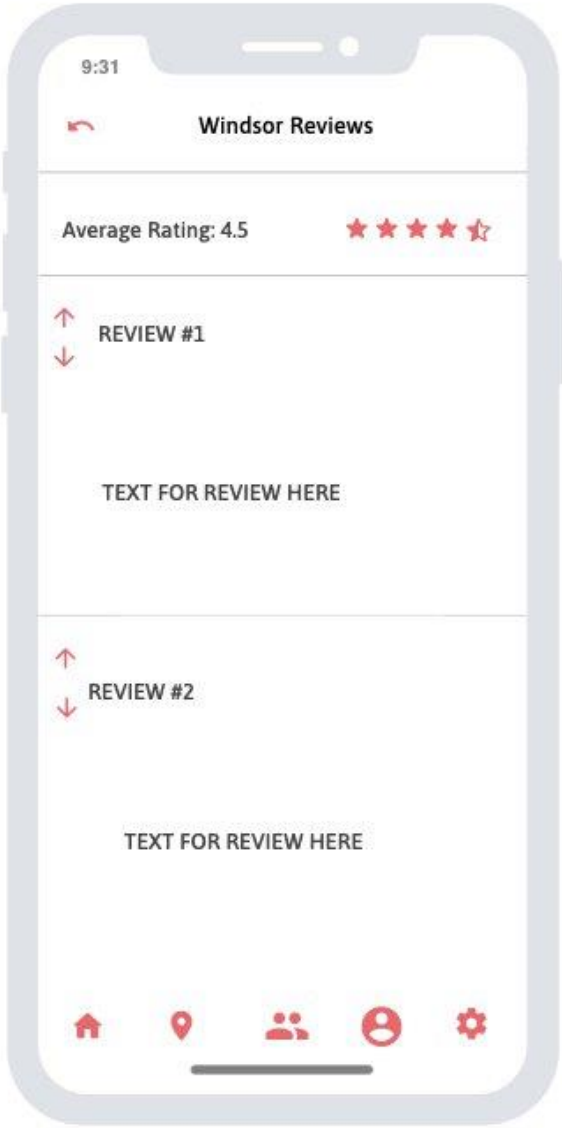
The Menus tab shows the selection of dining courts available to the user to choose from. The user will first select the desired court, and a new page will be generated that displays the list of menu items available in the dining court (next UI mockup).



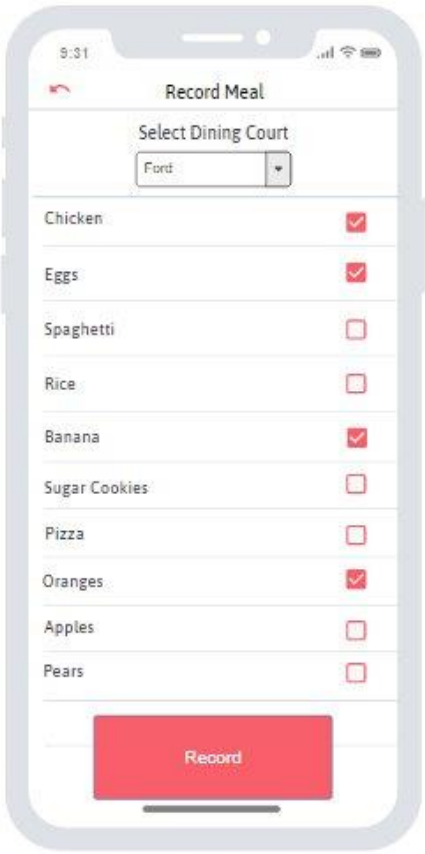
This sequential menu page displays the list of menu items available for the previously selected dining court. This page also reveals if a menu item is favorited by the user and associated allergy labels. The page also displays the aggregate user-submitted rating of the dining court.



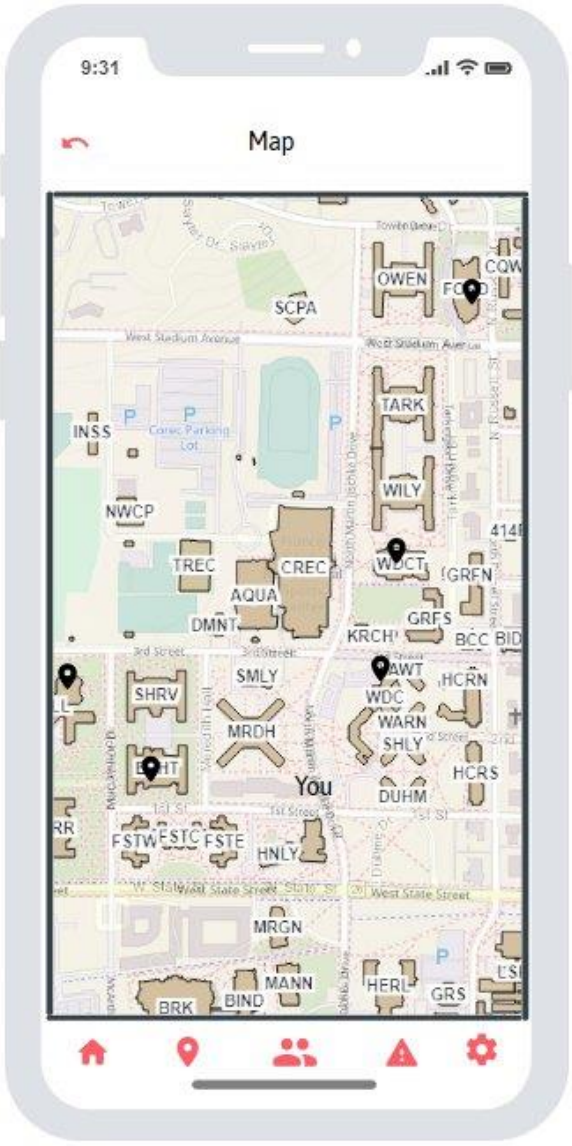
Selecting a menu item will generate the nutrition information page of the item. The standard nutrition information will be displayed here.



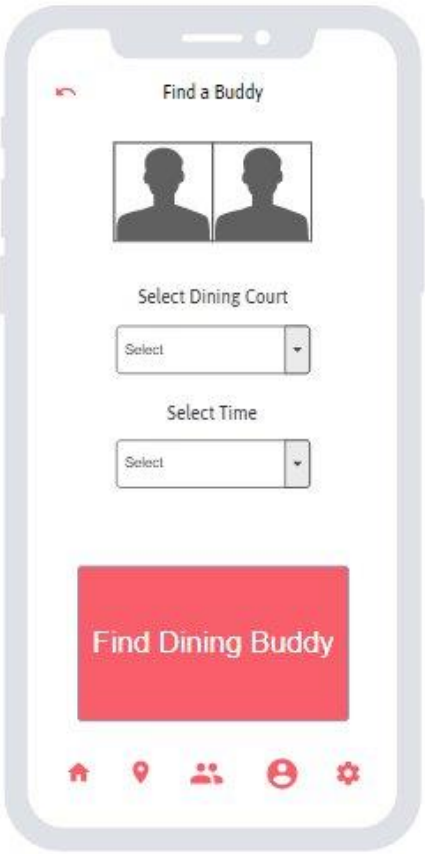
The Dining Court Reviews Page will display the average rating of users' reviews as well as the review text per user. Moreover, users can upvote or downvote other reviews.



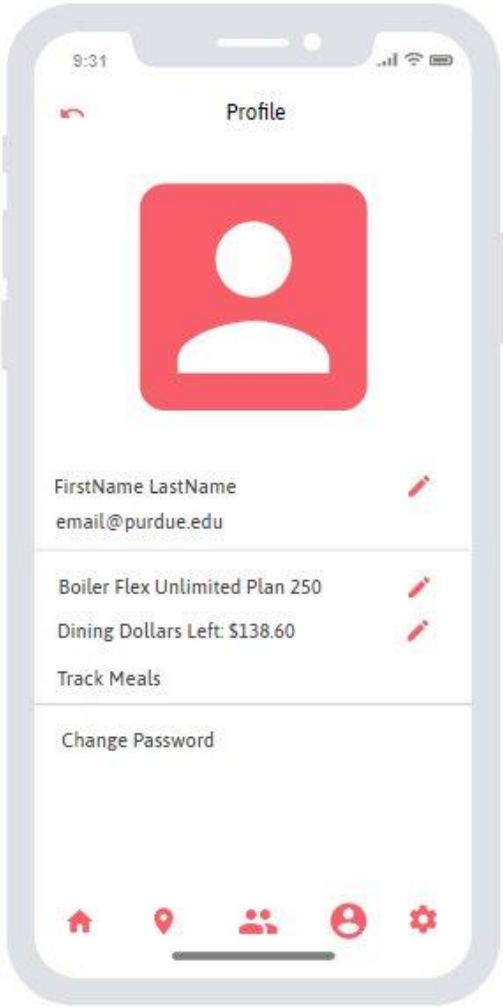
The Record Meal Page will allow the user to select all meal items consumed from a dining court. This information is used to personalize the meal recommendations for the user.



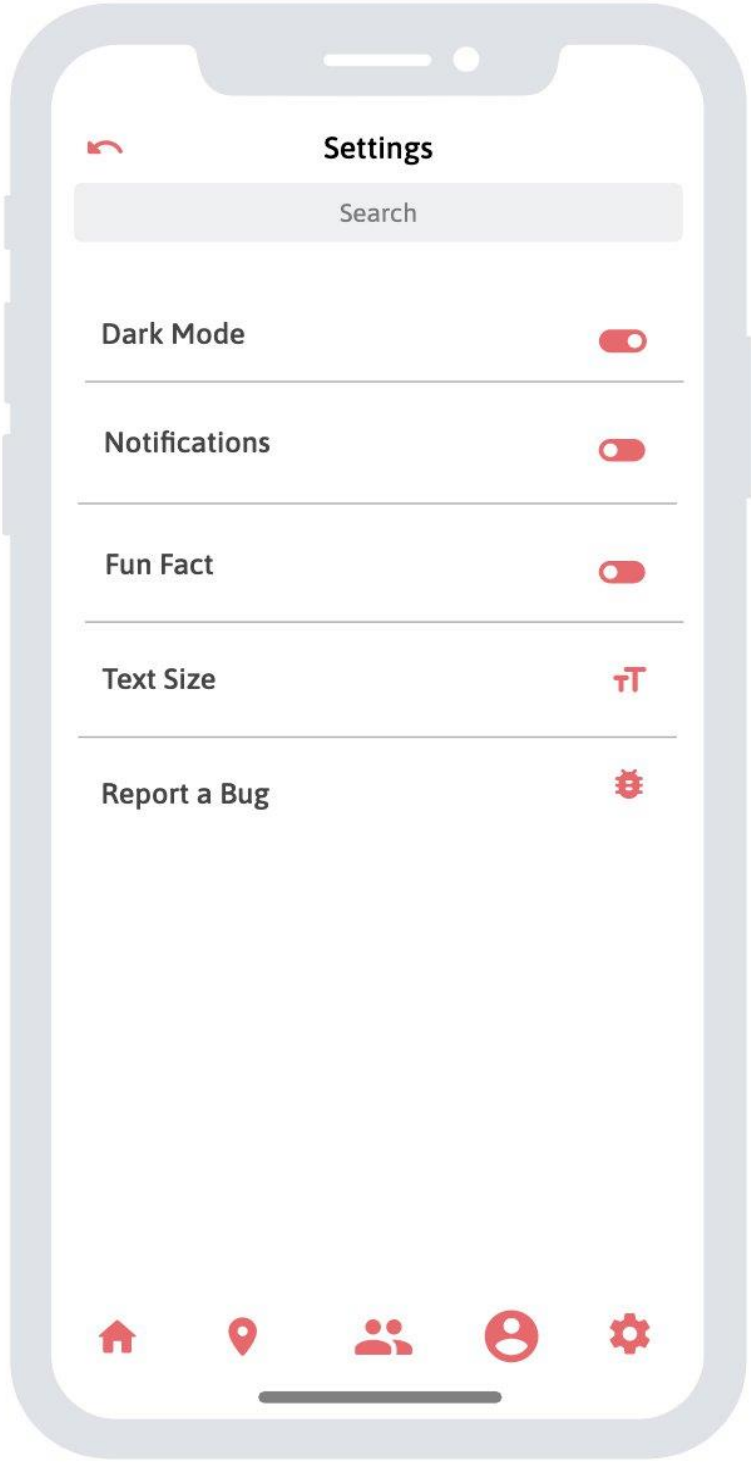
The Map page displays a map of the local Purdue area with the dining facilities pinned. The user's location will also be shown.



The Find a Buddy screen allows users to match with each other based on mutual dining court and time selections.



The profile page displays the user's personal information, dining dollar balance, and meal plan. The user can also change their password if they so choose.



The settings page allows users to customize their user-interface experience. This page additionally allows users to report a bug if any are found.