CourtSort Design Document

Team 12

Aahan Aggarwal, Carlo Cirrincione, William Eiffert, Karl Gaertner, Daniel Kambich, and Tyler Ruth.

Table of Contents

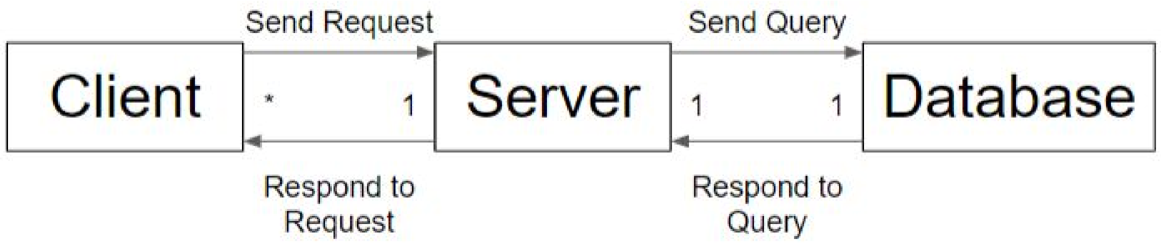
Purpose

The Purdue dining courts and menu application provide a fairly average experience. Whether it be getting lucky with tempura chicken or finding out Earhart's ice cream machine is broken, the overall quality can be hit or miss at times. Students may often have trouble deciding which dining court to eat at with their groups of friends, or be sitting alone without realizing their friends are eating across the dining court. The dining courts have potential to help students bond over a meal, while providing tasty food to replenish them after a long study session. Yet, the Purdue menu application's basic functionality does not satisfy these needs. In addition, third-party applications such as Yelp, Eat Out, and Google maps provide ratings for each restaurant, but not each dish, which matters when the dining courts feature different dishes of varying cuisine each day. Overall, these apps lack the necessary integration with the Purdue dining court system, leaving the problems unresolved.

We are designing a mobile application that crowdsources data such as dish ratings, equipment malfunctions, and the current capacity of dining courts in order to create a more social and enjoyable meal at the dining courts. We believe that CourtSort solves the problem of the mildly-disappointing dining court experience at Purdue and will help enhance the quality of life of our fellow Boilermakers.

Design Outline

This project will be a predictive dining court app where users can quickly find the best dining court to fit their preferences, needs, and social circles. Our application will use a client-server-database model to handle multiple clients requesting information from the database where we store all of our information on dining court food and group information and distributing the most up to date dining court data from the Purdue Dining API.



1. Client
    a. Where the user interacts with our system
    b. Will be a mobile application
    c. Will display all relevant information to the user such as dining court menus, chats, and notifications
    d. Will send requests to the server for updates on dining court food and events
2. Server
    a. Will handle communication between the client and the database
    b. Will send queries to the database for menu information, profiles, and friend groups
    c. Commands the database to add and update information on the user's actions
    d. Calculated overall food rating and food recommendations
    e. Will update the database with information from the Purdue Dining API
3. Database
    a. Stores all of the information for each user's profile, their friend groups, and dining court information
    b. Responds to queries sent by the server
    c. Typical data includes food ratings, group events, and profile information

A majority of our design choices revolved around user experience and creating an unintrusive, intuitive design. This ranged from the amount of notifications the user receives to the amount of functionality a user should have depending on their status. Additionally we had to consider how the specifics of certain systems should function such as determining if the user is in a dining court or what recommendations should be based on. Overall, we designed our system to be easy to use and to provide timely and relevant information to our users.

Design Issues

Functional Issues

How should it be determined that a user is currently in a dining court?

1. Track the user's location
2. Have the user manually "check-in"
3. Use a combination of the user's location at regular time intervals (30 minutes) and ask them to check-in if they are in the vicinity of a dining court

Choice and Explanation: We felt that this option provided the best balance between battery efficiency, accuracy, and convenience. We felt that solely relying on location data would be inefficient and for those who live in a residence hall above a dining court, the application would most likely be inaccurate. In addition, if users had to manually launch CourtSort to check-in every time they walked into a dining court, this core feature would probably be underutilized. Instead, by sending out a simple notification asking the user to confirm if they are in a dining court it would make it much quicker for the user and allow the app to be more effective.

How much functionality will be given to users who do not choose to sign in?

1. Full functionality such as being able to rate food, join groups, receive food recommendations
2. Basic functionality such as viewing dining court menus, operatings hours, and food ratings

Choice and Explanation: We chose to go with just giving users who do not sign in basic functionality because not all users would take advantage of the extra features we are providing and would simply like to know about various dishes served at dining courts and the general consensus on food options. In addition, some users might be conscious about their data and by not signing in so this gives those users a reason to use CourtSort.

What information will be needed to determine food recommendations?

1. Past ratings of dishes
2. Location data to determine what dining options are nearby
3. Current day of the week
4. Time of day
5. Friends' past food ratings
6. Dietary Restrictions
7. Friend's locations
8. Combination of options 1, 2, 6, and 7

Choice and Explanation: We felt that past ratings of dishes, location data, dietary restrictions, and friend's locations were probably the most important factors that we should keep track of in order to determine accurate food recommendations for an individual. We felt that some of the other options such as current day of the week, time of day, and friends' past food ratings really would not do a good job of representing a user's specific tastes and could get complicated with no additional benefit. Therefore, we felt that the options we chose (1, 2, 6, and 7) had a good balance of complexity and accurateness and wouldn't be overly difficult to implement.

How will food recommendations be handled and displayed in the application?

1. Three dining courts (for breakfast, lunch, and dinner) along with a list of top foods for each will be displayed once per day
2. Each time the user opens the application, a single dining court along with a list of top foods will be recommended based on parameters such as the user's location, past ratings of dishes, dietary restrictions, and friend's locations
3. There will be a section with a list of every dining court and the top picks of food for that specific dining court (based on parameters such as previous ratings, and dietary restrictions)
4. <u>Combination of options 2 and 3</u>

Choice and Explanation: We decided to go with option 4, which means we will have a dining court and food recommendation that is based on parameters such as the user's current location, friends' current locations, dietary restrictions, and previous dish ratings. In addition, we will also implement another section that contains a list of each dining court and its top foods with the option to change the date and meal type (breakfast, lunch, late lunch, or dinner). We felt that this was the best option because we found that there were 2 main ways people used the Purdue menu app. The first case is that people tend to check the menus ahead of time to plan out which dining court they want to eat at by comparing all other options. The second case is that some people launch the app when they want to eat very soon, which involves choosing a dining court that is relatively close and has the better food options out of the others they looked at. Thus, we felt that these two implementations would satisfy both cases.

How often will a user receive a food recommendation notification?

1. Once per day
2. <u>One before each meal, which a user can set preferences for how often and when they usually eat</u>

Choice and Explanation: We did not want to spam a user with more notifications than necessary, but, we also felt that having a notification before each meal would be appropriate and convenient. Therefore, we decided that a notification should be sent a notification before there preferred meal times (if the user configures it).

How will we able to accurately determine the busyness of dining court?

1. Use the API to get information about how many people are currently in line
2. Use the dining court "check-in" information to determine how many people who are using CourtSort are currently at each dining court
3. <u>Combination of options 1 and 2</u>

Choice and Explanation: We decided to determine busyness of a dining court by utilizing both the API and by seeing how many CourtSort users have "checked-in" to a specific dining court. We felt that this would help give the most accurate judgement of busyness, because the CourtSort check-in feature only shows how many CourtSort users are in a given dining court and the API does not have the most real-time estimate on how long the line is. By merging the two it should give us a better estimate on how many people are currently in a dining court.

How should groups create plans to eat?

1. One user can create an event to go eat, along with choosing the dining court and the time
2. One user can create an event to go eat, however, the group votes on the dining court and the time
3. One user can create an event to go eat, however, the group votes on the time, while the application recommends a dining court

Choice and Explanation: We decided that when groups create plans/events to go eat that each individual in the group has a chance to vote on the time, however, CourtSort recommends the dining court. We chose this option because we felt that it was an optimal combination of simplicity and customization. We also felt that the goal of CourtSort was to help people decide where to eat, so this option made the most sense because it takes away stress from the group and helps make the difficult decision easier, while also giving the group the flexibility to vote on the most appropriate time for everyone. In the process of making this decision we did bring up the case that if one individual had a very strong dining court preference, how would the group recommendation turn out. However, we decided if that were the case that it should be handled outside CourtSort as we desire the group recommendation to be simple and easy, where as that would add too much complexity.

How should individual friends in comparison to friend groups be distinguished and displayed when messaging?

1. Friends and friend groups show up in the same list and are essentially treated the same
2. Friends have a separate list from groups and are managed separately

Choice and Explanation: We decided to treat friends and friend groups similarly by having them both show up in the same list. We decided to go with this option because we felt that having a friends list separate from a groups list was overly complicated distinction for roughly the same functionality. In addition, we felt that the user would be most familiar with it organized with both friends and groups together in the same list since that is how messaging apps such as Snapchat, Whatsapp, and other messaging clients have it organized.

Non-Functional Issues

What technologies will we use to develop our back-end?

1. Firebase
2. Node.js server with a MongoDB database

Choice and Explanation: We chose to go with Firebase for our back-end. We selected this option because the majority of our team has very little experience with databases and back-end development. We felt that trying to connect the Node.js server with the MongoDB database, while making it secure might take extra time where as Firebase provides tools for Hosting, Authentication, and a secure Realtime Database in one complete package. As a result, we determined Firebase was the simplest and quickest option to develop the back-end given our lack of experience, which results in more time to focus on the features such as the food recommendation algorithm.

How will dining court information be obtained by the Purdue Dining API?

1. Each client makes requests directly to the Purdue Dining API

2. <u>The server makes requests to the Purdue Dining API and then the client makes requests to the server when loading dining court information</u>

Choice and Explanation: We have chosen to go with option 2 where the server makes the requests to the API rather than the client. This is because the client should not display the raw menu information directly from the API, instead the client must display the menu along with the ratings for each dish and personalized food recommendations. Thus, the server will need to have the menu stored in the database along with the ratings so that it can perform food recommendations for a given user, as well as be able to send that information to the client when requested.

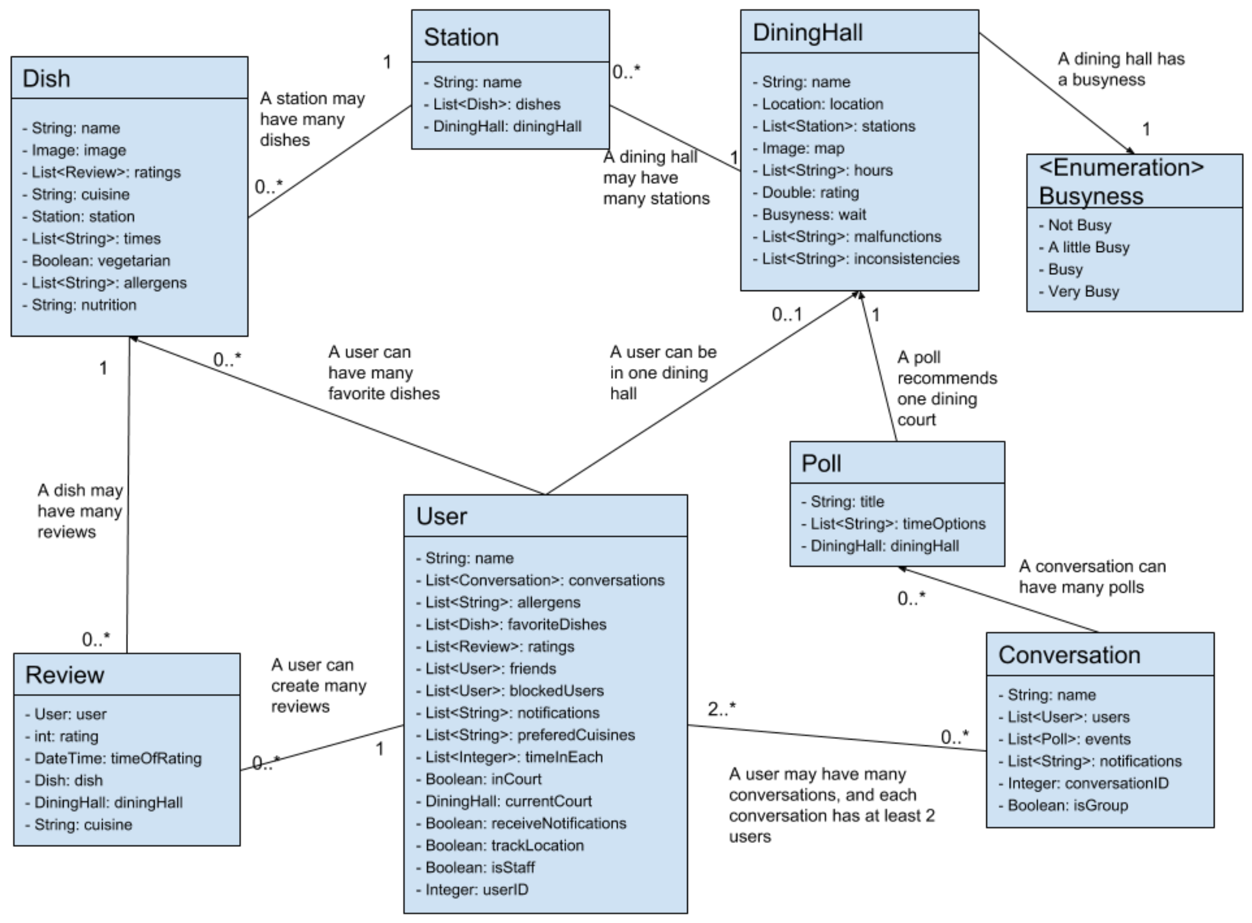
What technologies will we use to develop our front-end?

1. Android Studio (Java)
2. <u>React Native (Javascript)</u>
3. Xcode (Swift)

Choice and Explanation: We chose to implement our front-end using Javascript with React Native. The reason we chose this option is because we felt that React Native would be the simplest to learn for those of us on the team that do not have much experience with mobile development. In addition, many of us have some experience with web development which React Native is similar to. Moreover, React Native allows us to develop for iOS and Android at the same time, thus increasing the availability of our app and the amount of potential users we can attain.

Design Details

Class Diagram


Station
- String: name
- List<Dish>: dishes
- DiningHall: diningHall
DiningHall
- String: name
- Location: location
- List<Station>: stations
- Image: map
- List<String>: hours
- Double: rating
- Busyness: wait
- List<String>: malfunctions
- List<String>: inconsistencies
Dish
- String: name
- Image: image
- List<Review>: ratings
- String: cuisine
- Station: station
- List<String>: times
- Boolean: vegetarian
- List<String>: allergens
- String: nutrition
<Enumeration> Busyness
- Not Busy
- A little Busy
- Busy
- Very Busy
A station may have many dishes
1
0..*
0..*
1
A dining hall may have many stations
A dining hall has a busyness
1
0..1
1
A user can have many favorite dishes
0..*
1
A user can be in one dining hall
A poll recommends one dining court
A dish may have many reviews
0..*
Poll
- String: title
- List<String>: timeOptions
- DiningHall: diningHall
A conversation can have many polls
0..*
User
- String: name
- List<Conversation>: conversations
- List<String>: allergens
- List<Dish>: favoriteDishes
- List<Review>: ratings
- List<User>: friends
- List<User>: blockedUsers
- List<String>: notifications
- List<String>: preferedCuisines
- List<Integer>: timeInEach
- Boolean: inCourt
- DiningHall: currentCourt
- Boolean: receiveNotifications
- Boolean: trackLocation
- Boolean: isStaff
- Integer: userID
Review
- User: user
- int: rating
- DateTime: timeOfRating
- Dish: dish
- DiningHall: diningHall
- String: cuisine
A user can create many reviews
0..*
1
2..*
0..*
A user may have many conversations, and each conversation has at least 2 users
Conversation
- String: name
- List<User>: users
- List<Poll>: events
- List<String>: notifications
- Integer: conversationID
- Boolean: isGroup

Description of Classes

Dish

- The dish is the lowest level object, and is how we will store each individual dish in the menu.
- It will be created every time a brand new dish is created at a station in a dining hall.
- It contains past information of votes for each time it was served, as well as nutritional and allergenic facts.

DiningHall

- The DiningHall object is created when we initialize the database and will be fixed with a certain number of stations, which themselves will contain their dishes.
- This can be updated regularly with opening times and how busy it is based on user votes.
- It also contains user reported malfunctions and inconsistencies.

Busyness

- This is a simple enum which contains values for how busy a dining court is.
- Users are allowed to vote only according to these options.

User

- The user object is the main object created every time a new user signs up using any method.
- It will have information on all their friends, conversations, nutritional and allergen information, and privacy related issues such as location tracking.
- It will also have a list of every time they have rated a particular dish, which allows us to make intelligent predictions.
- Lastly, it also contains variables to track whether they are in a particular dining court currently, helping us know when they are checked in and can vote on dishes.

Station

- This object will also be created when we initialize the database.
- It will only contain a refreshing list of all its dishes, as any other information will be in its parent class the DiningHall.

Conversation

- Each conversation between users will have its own object.
- Both groups and conversations between 2 people will be handled by the same object, with a boolean variable to differentiate.
- Will contain all previous and current events between the members of the conversation.

Poll

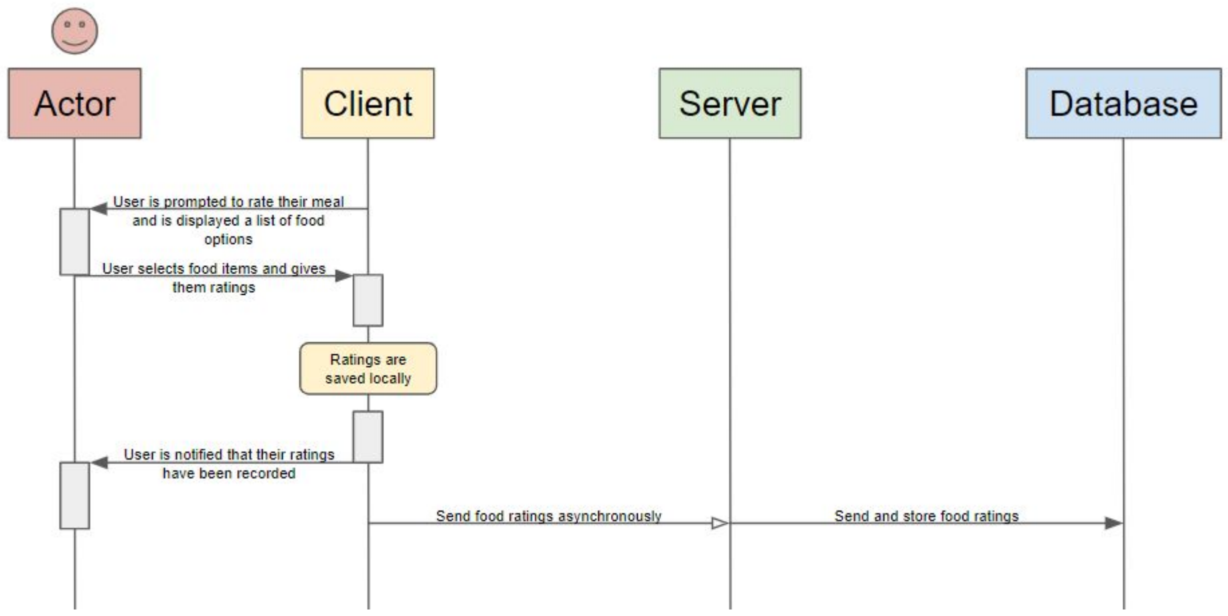
- A poll is an object that helps in creating a basic event within a conversation.
- This will allow users to choose the time they prefer.

Review

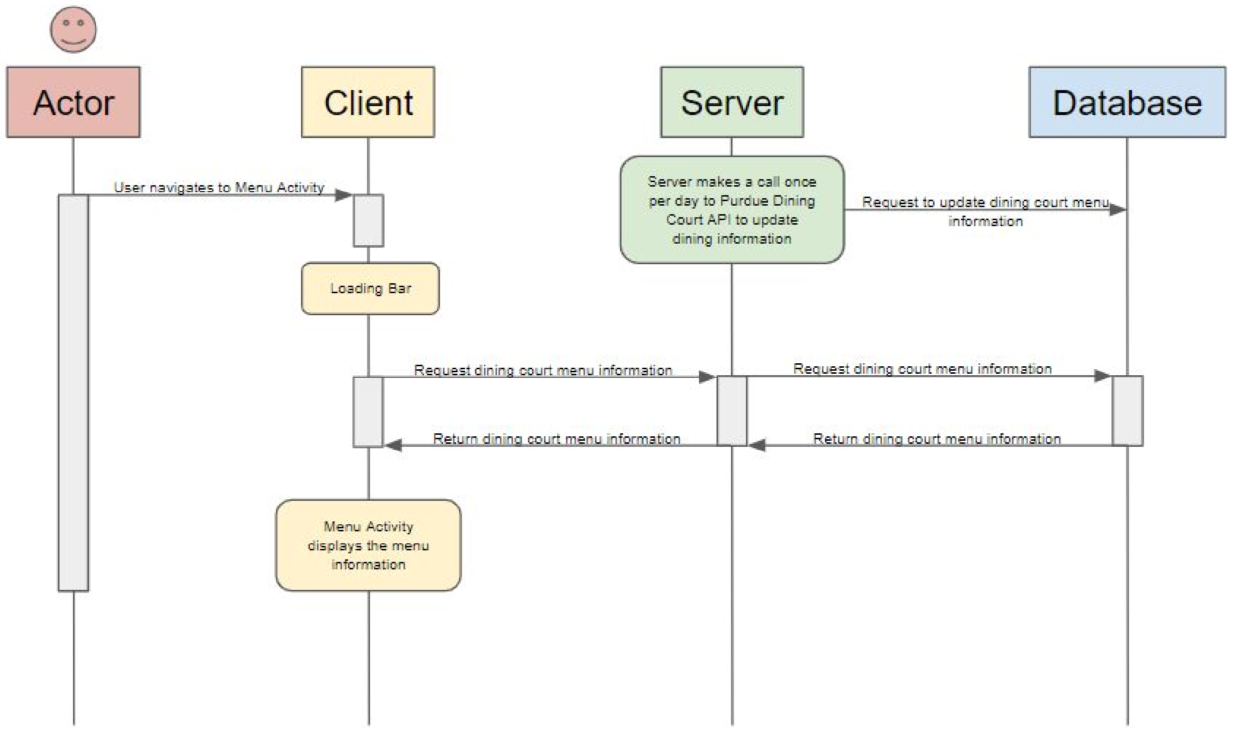
- This is an object created every time a user marks a review for any item.
- It contains all the information about the dish and where it was.

Sequence Diagrams

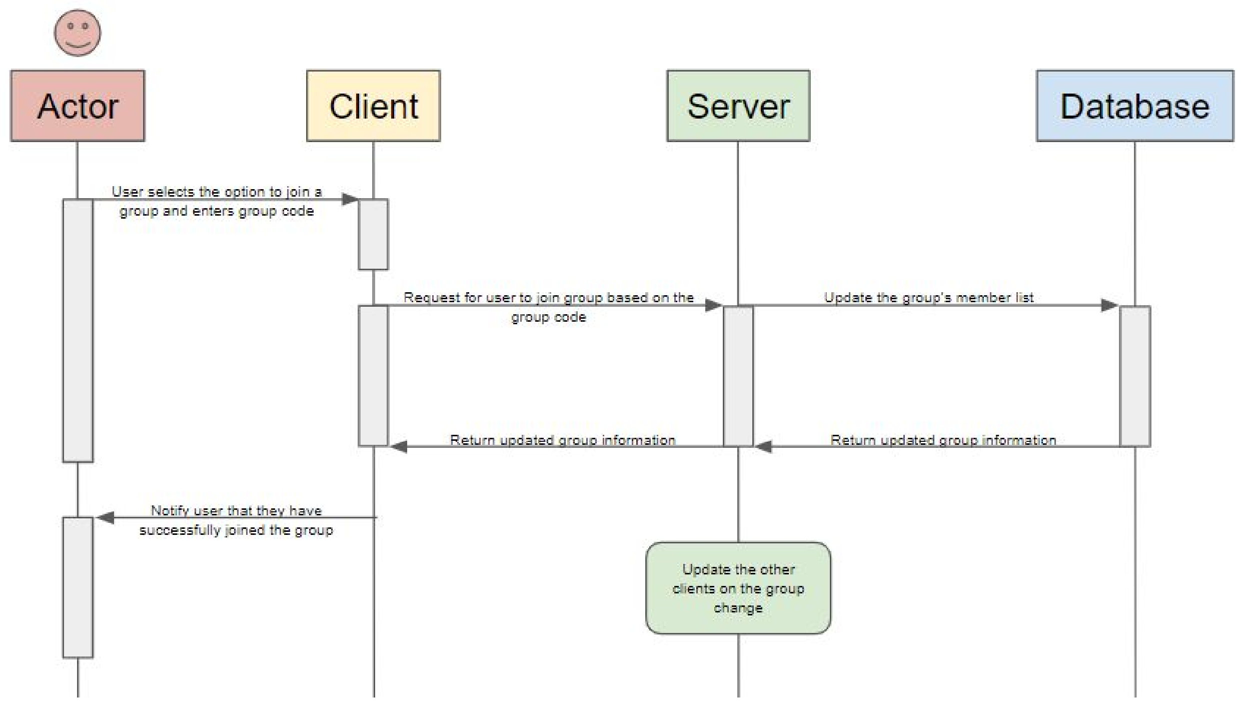
Sequence of events for rating a meal


Actor
Client
Server
Database
User is prompted to rate their meal and is displayed a list of food options
User selects food items and gives them ratings
Ratings are saved locally
User is notified that their ratings have been recorded
Send food ratings asynchronously
Send and store food ratings

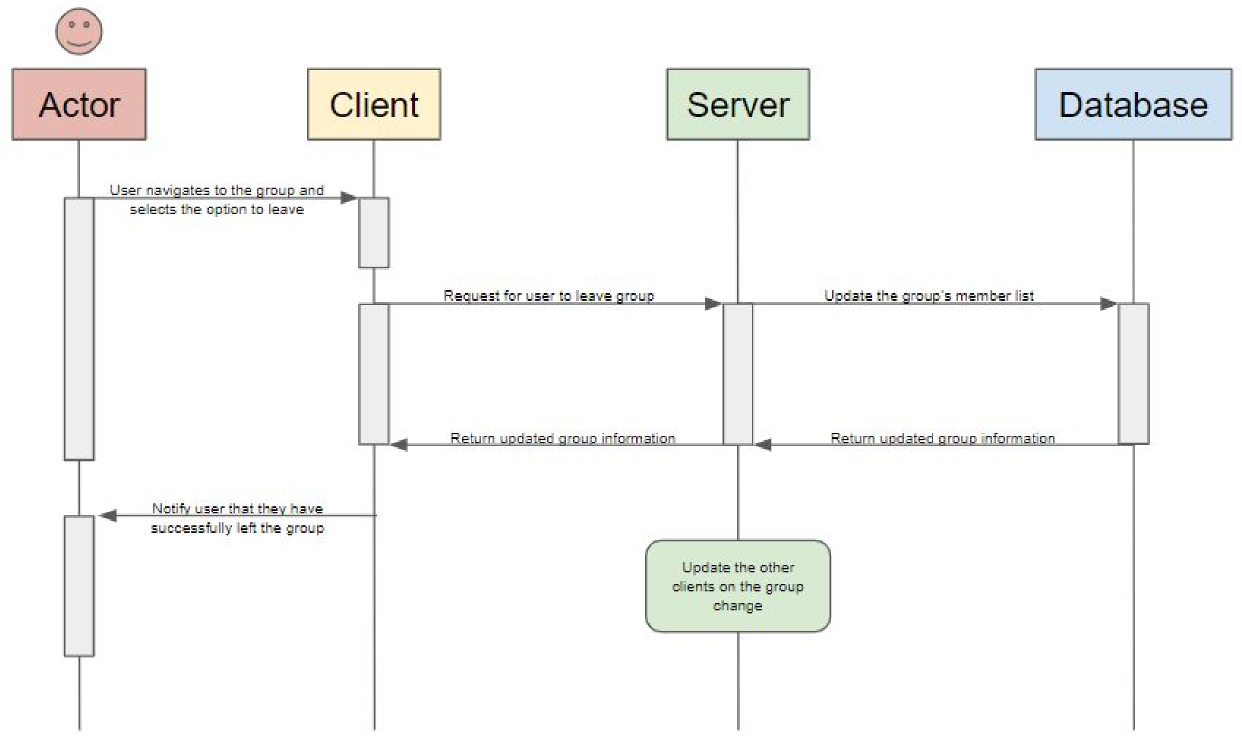
Sequence of events when user requests dining court menus


Actor
Client
Server
Database
User navigates to Menu Activity
Server makes a call once per day to Purdue Dining Court API to update dining information
Request to update dining court menu information
Loading Bar
Request dining court menu information
Request dining court menu information
Return dining court menu information
Return dining court menu information
Menu Activity displays the menu information

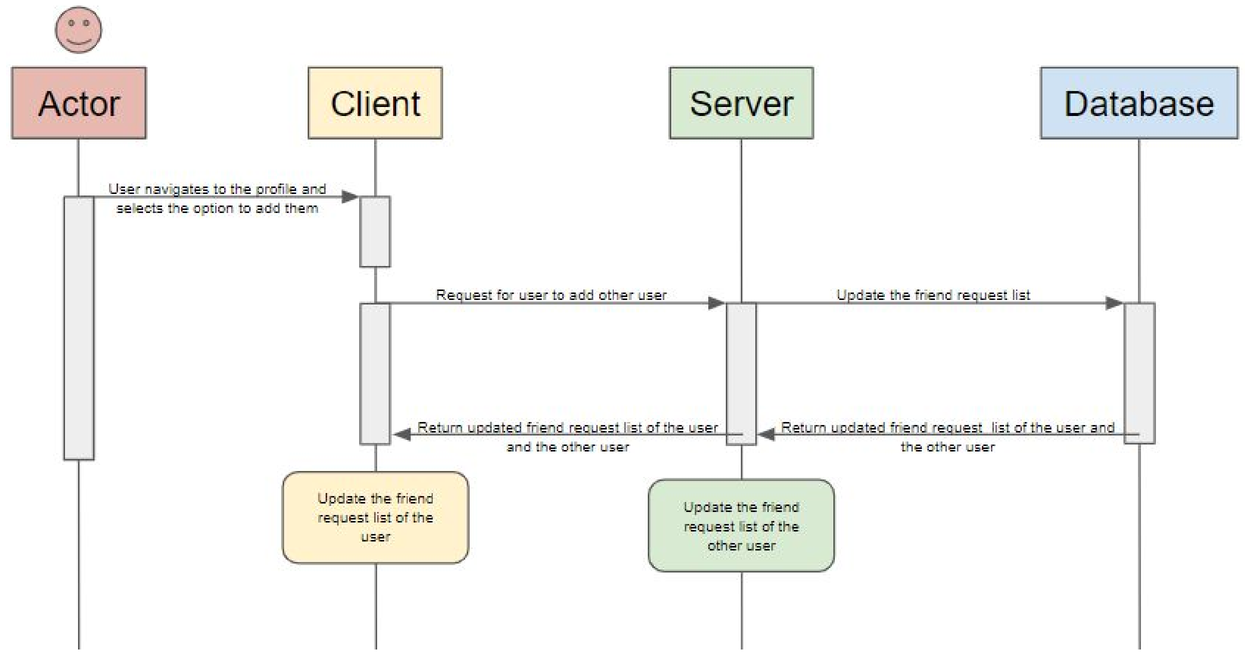
Sequence of events when a user joins a group


Actor
Client
Server
Database
User selects the option to join a group and enters group code
Request for user to join group based on the group code
Update the group's member list
Return updated group information
Return updated group information
Notify user that they have successfully joined the group
Update the other clients on the group change

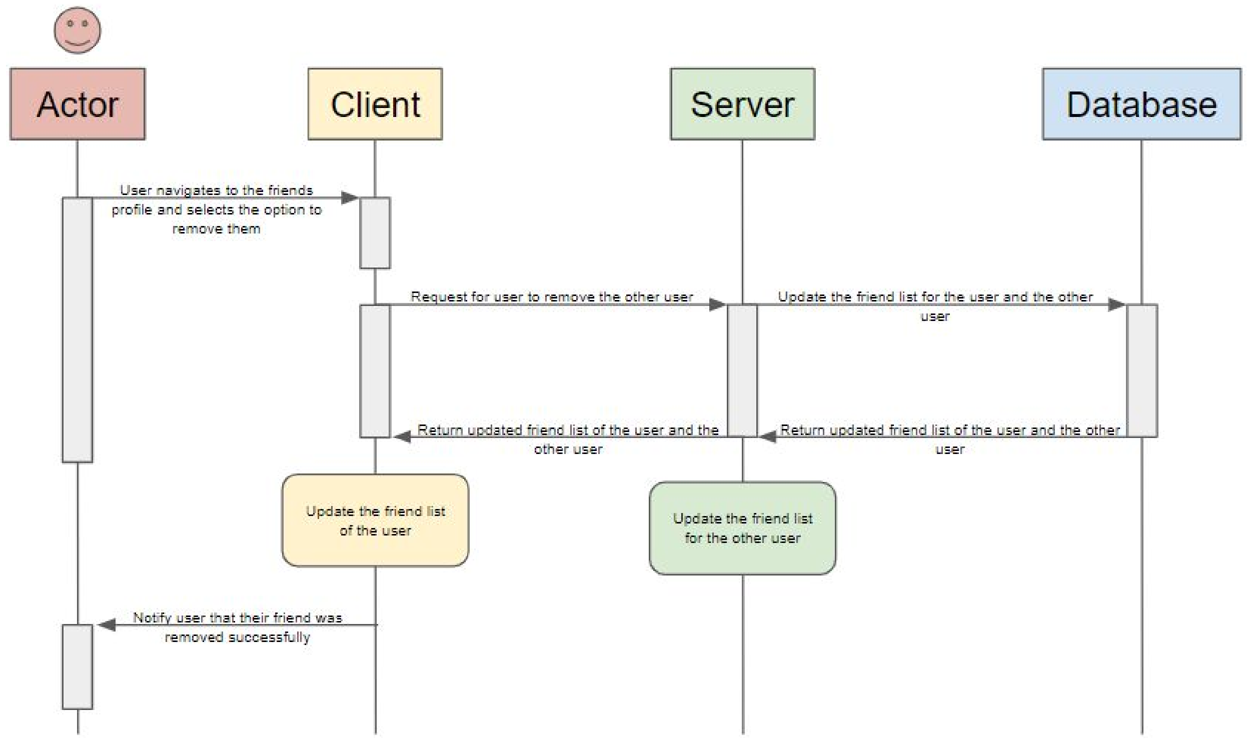
Sequence of events when a user leaves a group


Actor
Client
Server
Database
User navigates to the group and selects the option to leave
Request for user to leave group
Update the group's member list
Return updated group information
Return updated group information
Notify user that they have successfully left the group
Update the other clients on the group change

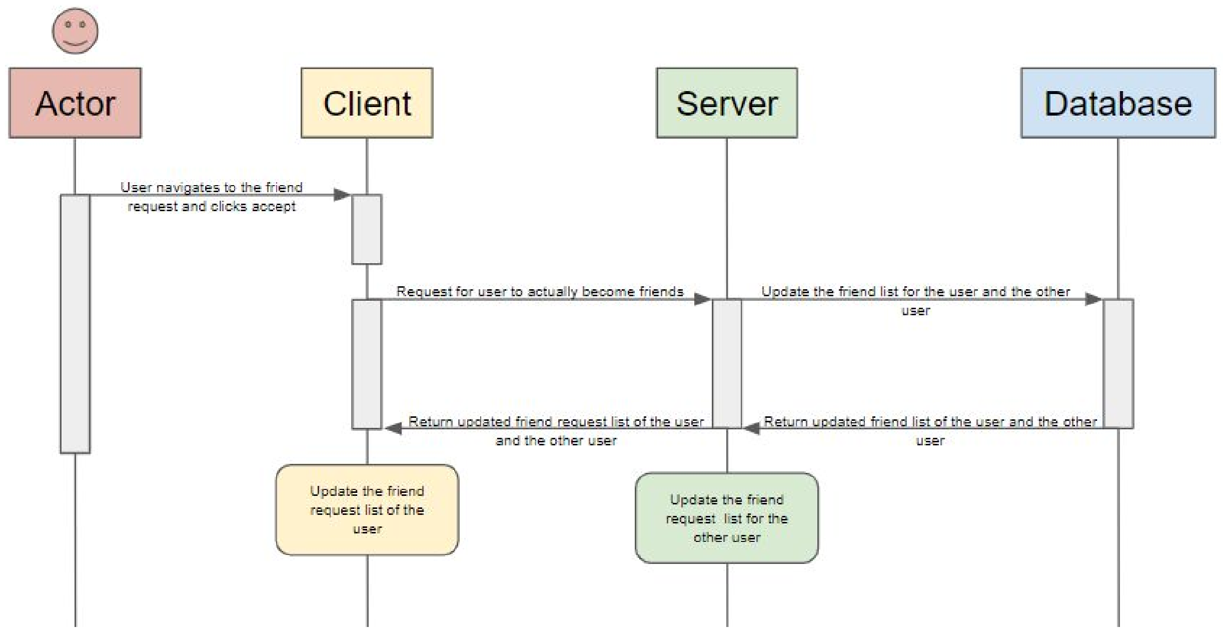
Sequence of events when a user adds a friend


Actor
Client
Server
Database
User navigates to the profile and selects the option to add them
Request for user to add other user
Update the friend request list
Return updated friend request list of the user and the other user
Return updated friend request list of the user and the other user
Update the friend request list of the user
Update the friend request list of the other user

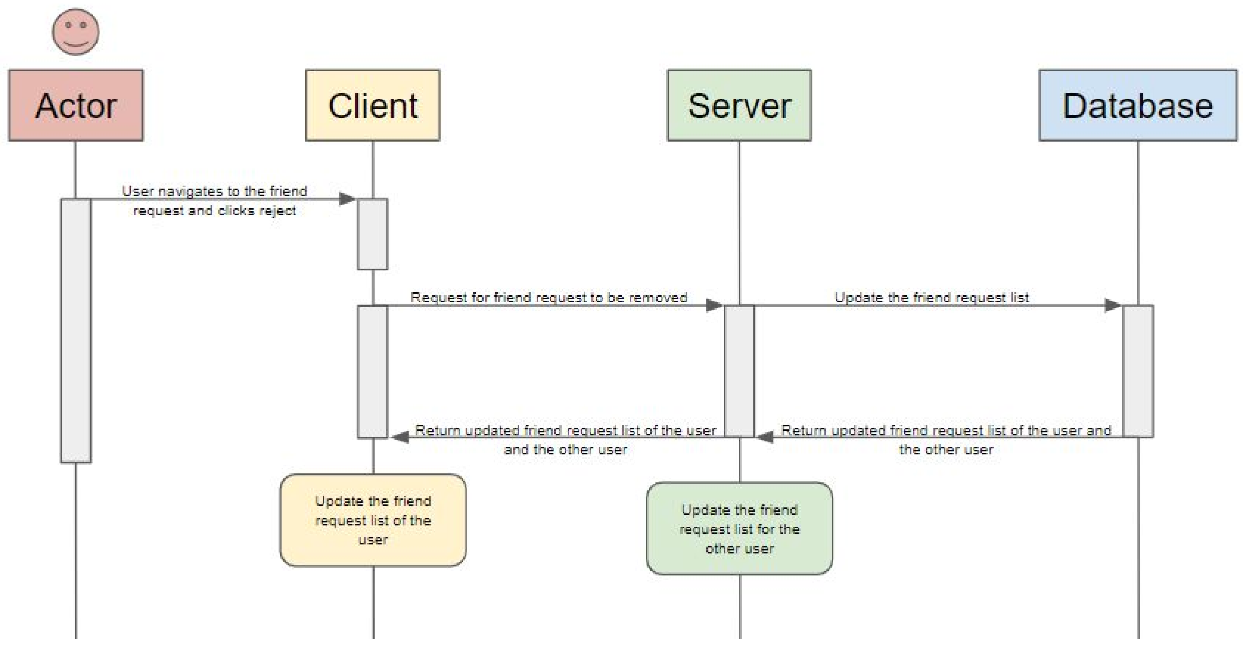
Sequence of events when a user removes a friend


Actor
Client
Server
Database
User navigates to the friends profile and selects the option to remove them
Request for user to remove the other user
Update the friend list for the user and the other user
Return updated friend list of the user and the other user
Return updated friend list of the user and the other user
Update the friend list of the user
Update the friend list for the other user
Notify user that their friend was removed successfully

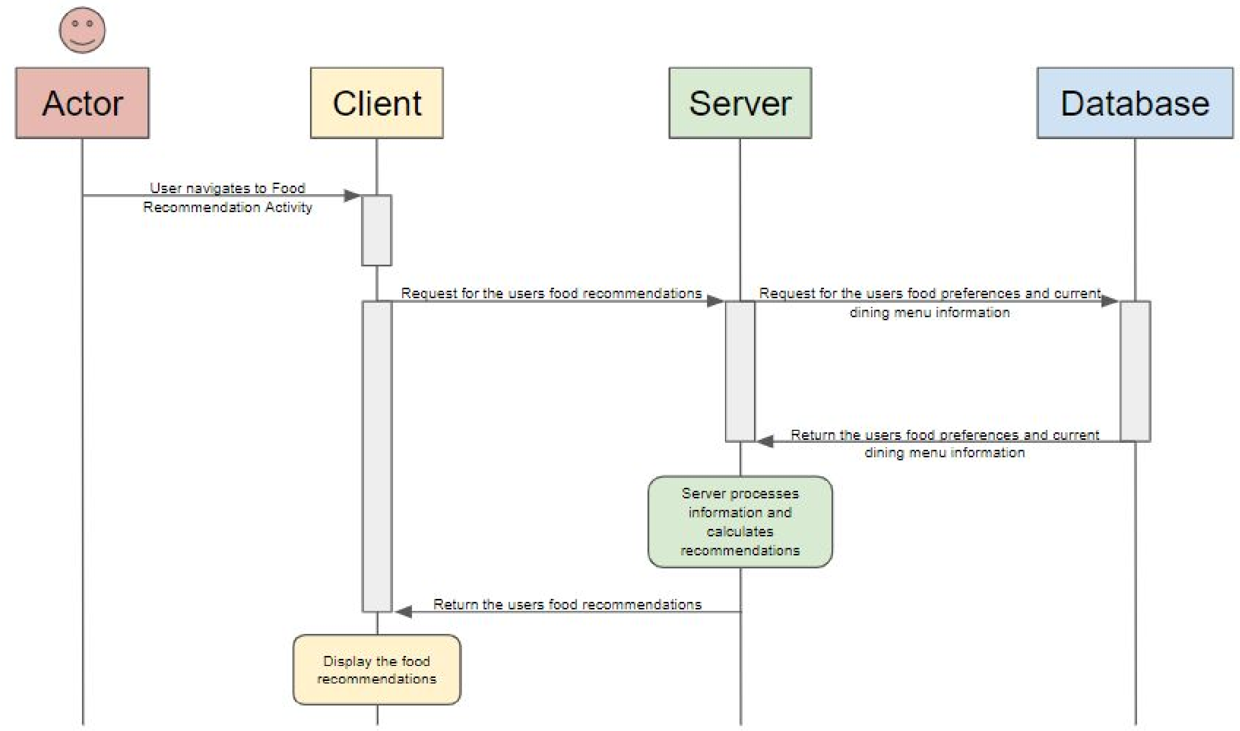
Sequence of events when a user accepts a friend request


Actor
Client
Server
Database
User navigates to the friend request and clicks accept
Request for user to actually become friends
Update the friend list for the user and the other user
Return updated friend request list of the user and the other user
Return updated friend list of the user and the other user
Update the friend request list of the user
Update the friend request list for the other user

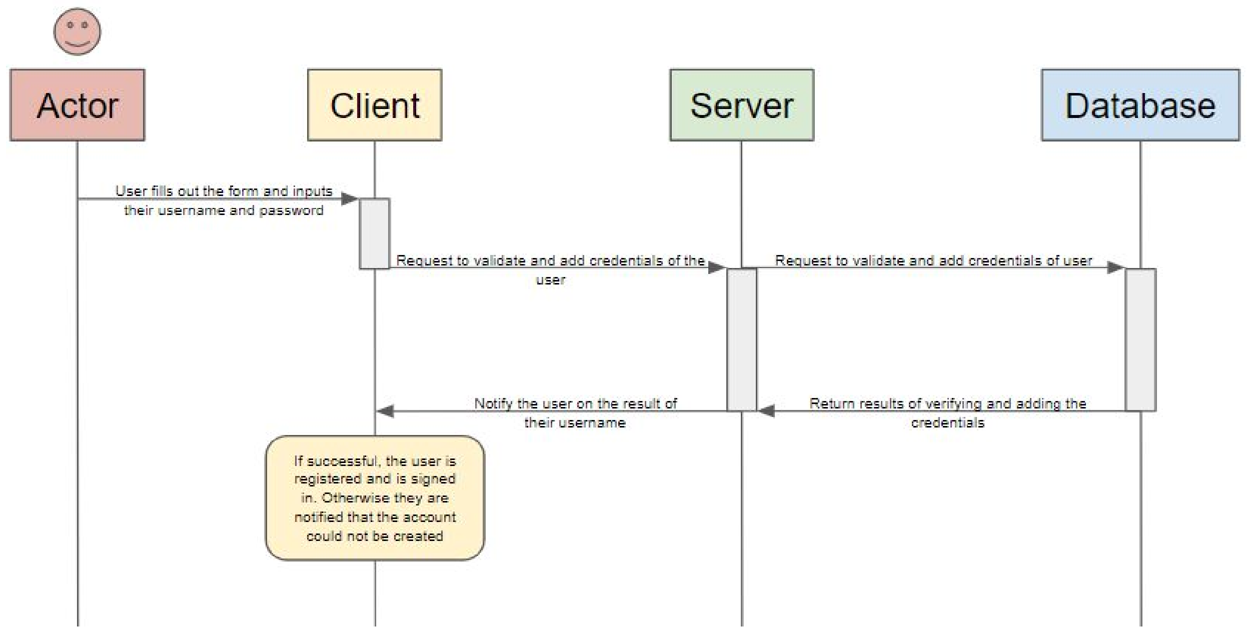
Sequence of events when a user declines a friend request


Actor
Client
Server
Database
User navigates to the friend request and clicks reject
Request for friend request to be removed
Update the friend request list
Return updated friend request list of the user and the other user
Return updated friend request list of the user and the other user
Update the friend request list of the user
Update the friend request list for the other user

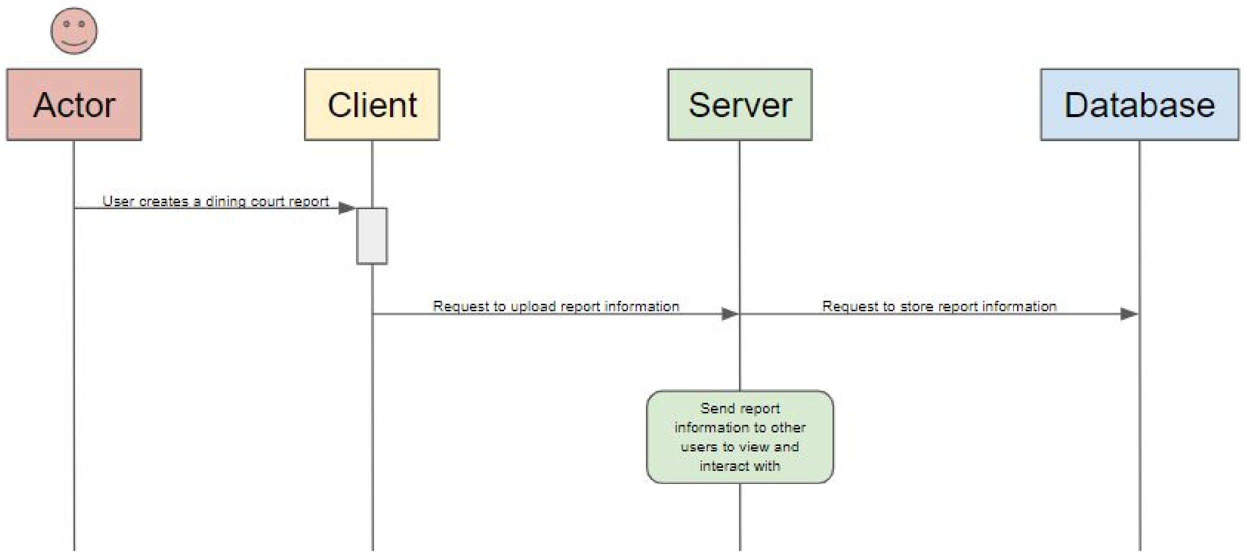
Sequence of events to view food recommendations


Actor
Client
Server
Database
User navigates to Food Recommendation Activity
Request for the users food recommendations
Request for the users food preferences and current dining menu information
Return the users food preferences and current dining menu information
Server processes information and calculates recommendations
Return the users food recommendations
Display the food recommendations

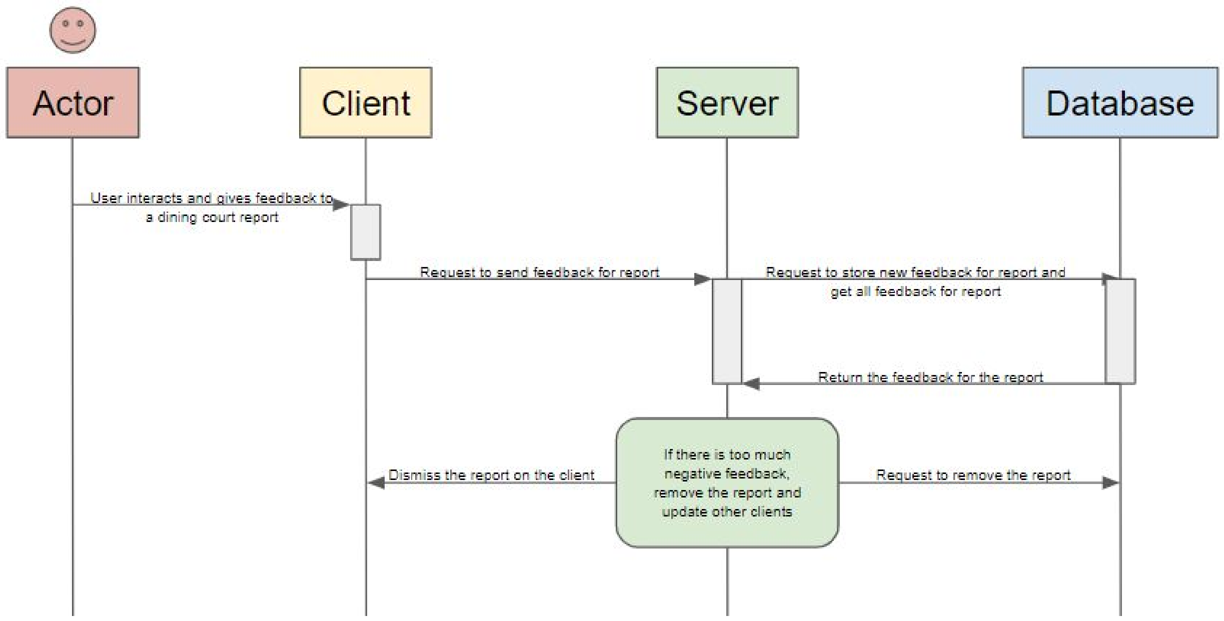
Sequence of events when creating a new account


Actor
Client
Server
Database
User fills out the form and inputs their username and password
Request to validate and add credentials of the user
Request to validate and add credentials of user
Return results of verifying and adding the credentials
Notify the user on the result of their username
If successful, the user is registered and is signed in. Otherwise they are notified that the account could not be created

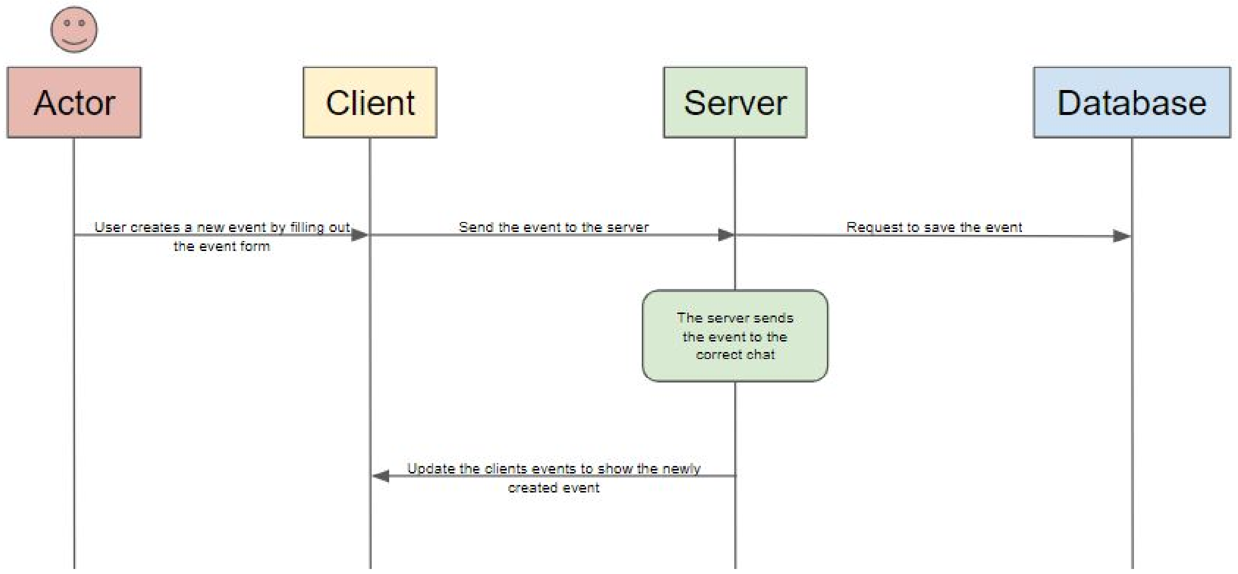
Sequence of events for making dining court reports


Actor
Client
Server
Database
User creates a dining court report
Request to upload report information
Request to store report information
Send report information to other users to view and interact with

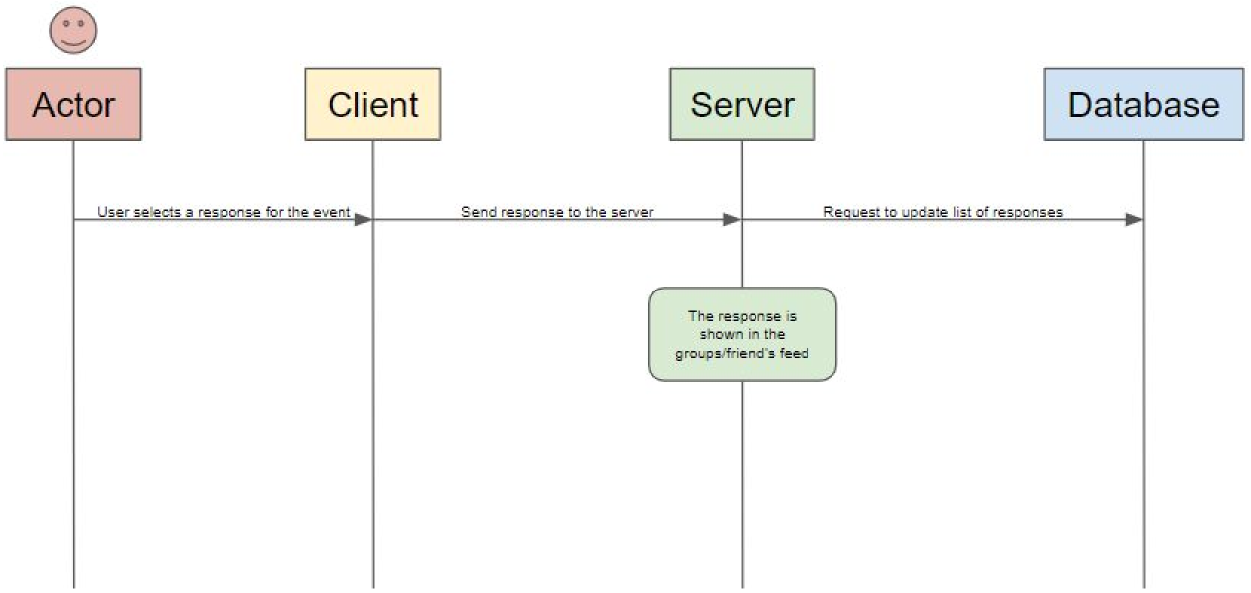
Sequence of events for interacting with dining court reports


Actor
Client
Server
Database
User interacts and gives feedback to a dining court report
Request to send feedback for report
Request to store new feedback for report and get all feedback for report
Return the feedback for the report
Dismiss the report on the client
If there is too much negative feedback, remove the report and update other clients
Request to remove the report

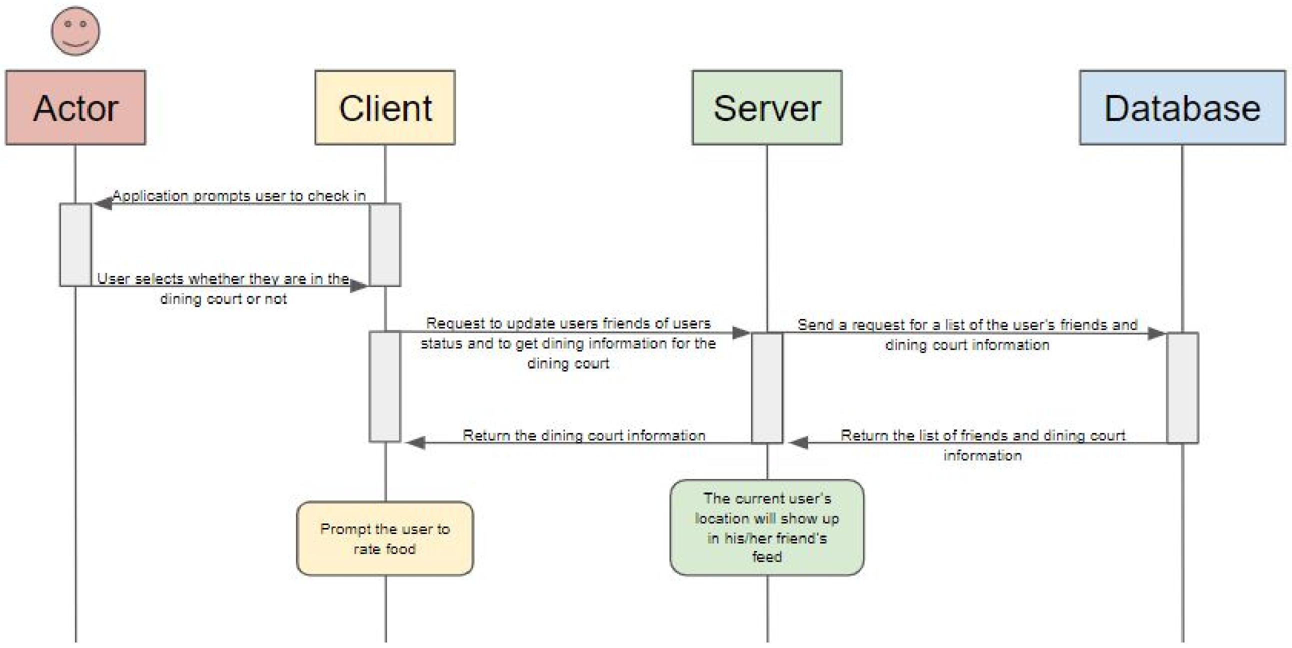
Sequence of events for creating an event


Actor
Client
Server
Database
User creates a new event by filling out the event form
Send the event to the server
Request to save the event
The server sends the event to the correct chat
Update the clients events to show the newly created event

Sequence of events for responding to an event

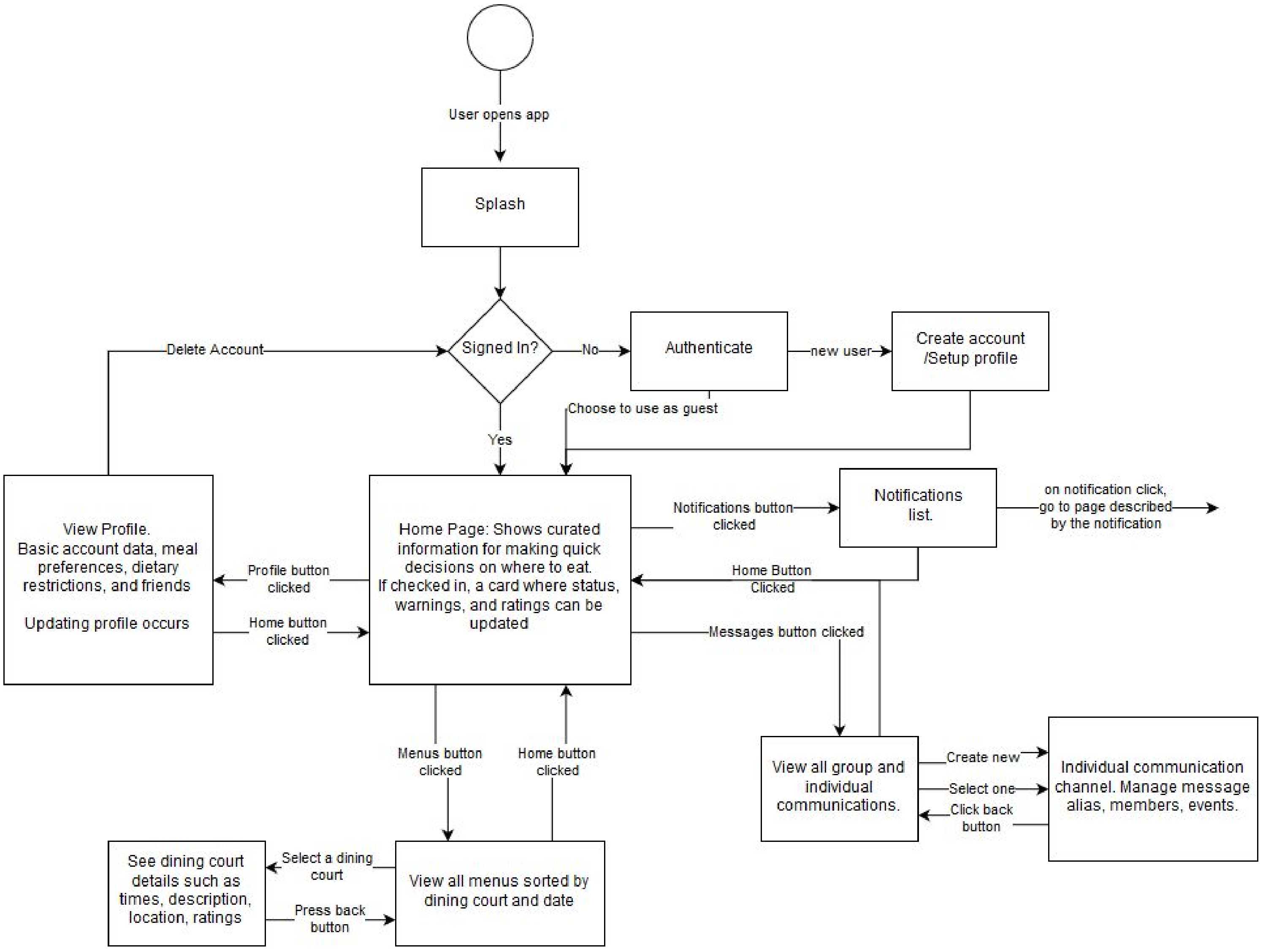

Actor
Client
Server
Database
User selects a response for the event
Send response to the server
Request to update list of responses
The response is shown in the groups/friend's feed

Sequence of events for checking into a dining court


Actor
Client
Server
Database
Application prompts user to check in
User selects whether they are in the dining court or not
Request to update users friends of users status and to get dining information for the dining court
Send a request for a list of the user's friends and dining court information
Return the dining court information
Return the list of friends and dining court information
Prompt the user to rate food
The current user's location will show up in his/her friend's feed

State Diagram

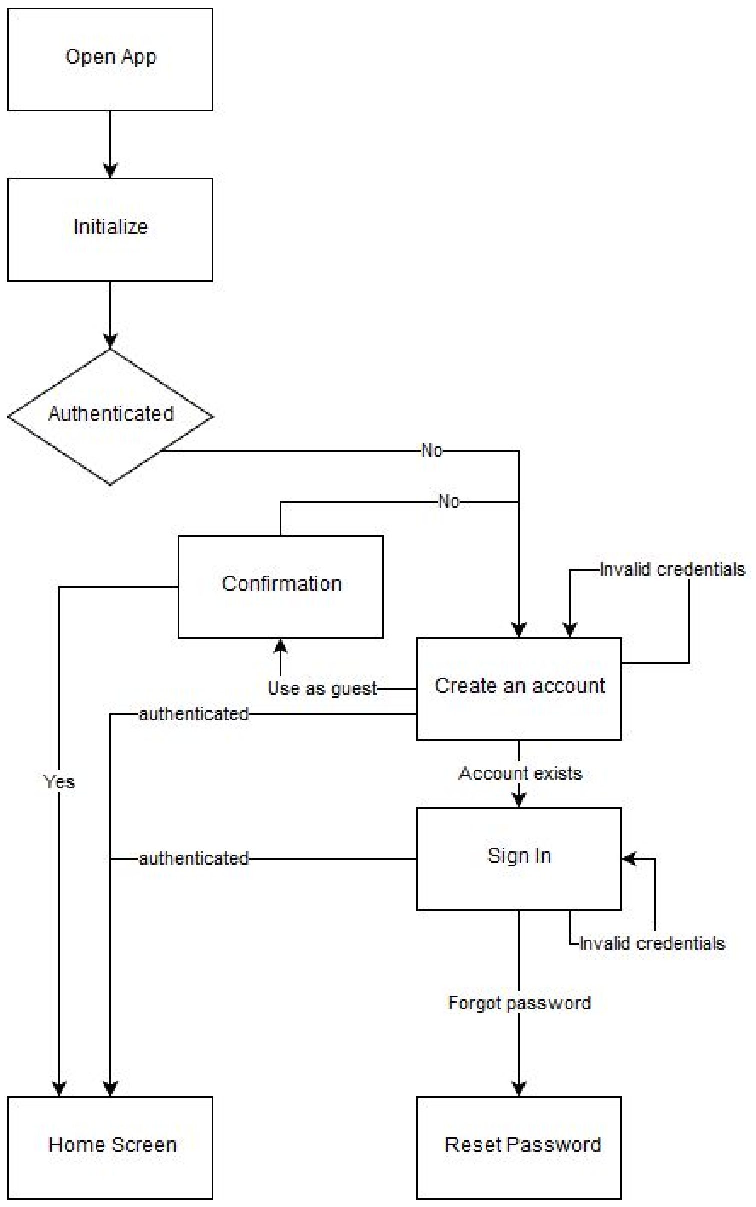
For sake of readability, any button click navigation to or from the home screen can be selected from each page, except for during authentication.


User opens app
Splash
Signed In?
No
Authenticate
new user
Create account /Setup profile
Choose to use as guest
Yes
Delete Account
View Profile. Basic account data, meal preferences, dietary restrictions, and friends
Updating profile occurs
Profile button clicked
Home button clicked
Home Page: Shows curated information for making quick decisions on where to eat. If checked in, a card where status, warnings, and ratings can be updated
Notifications button clicked
Notifications list.
on notification click, go to page described by the notification
Home Button Clicked
Messages button clicked
View all group and individual communications.
Create new
Select one
Click back button
Individual communication channel. Manage message alias, members, events.
Menus button clicked
Home button clicked
View all menus sorted by dining court and date
Select a dining court
Press back button
See dining court details such as times, description, location, ratings

Navigation Diagrams

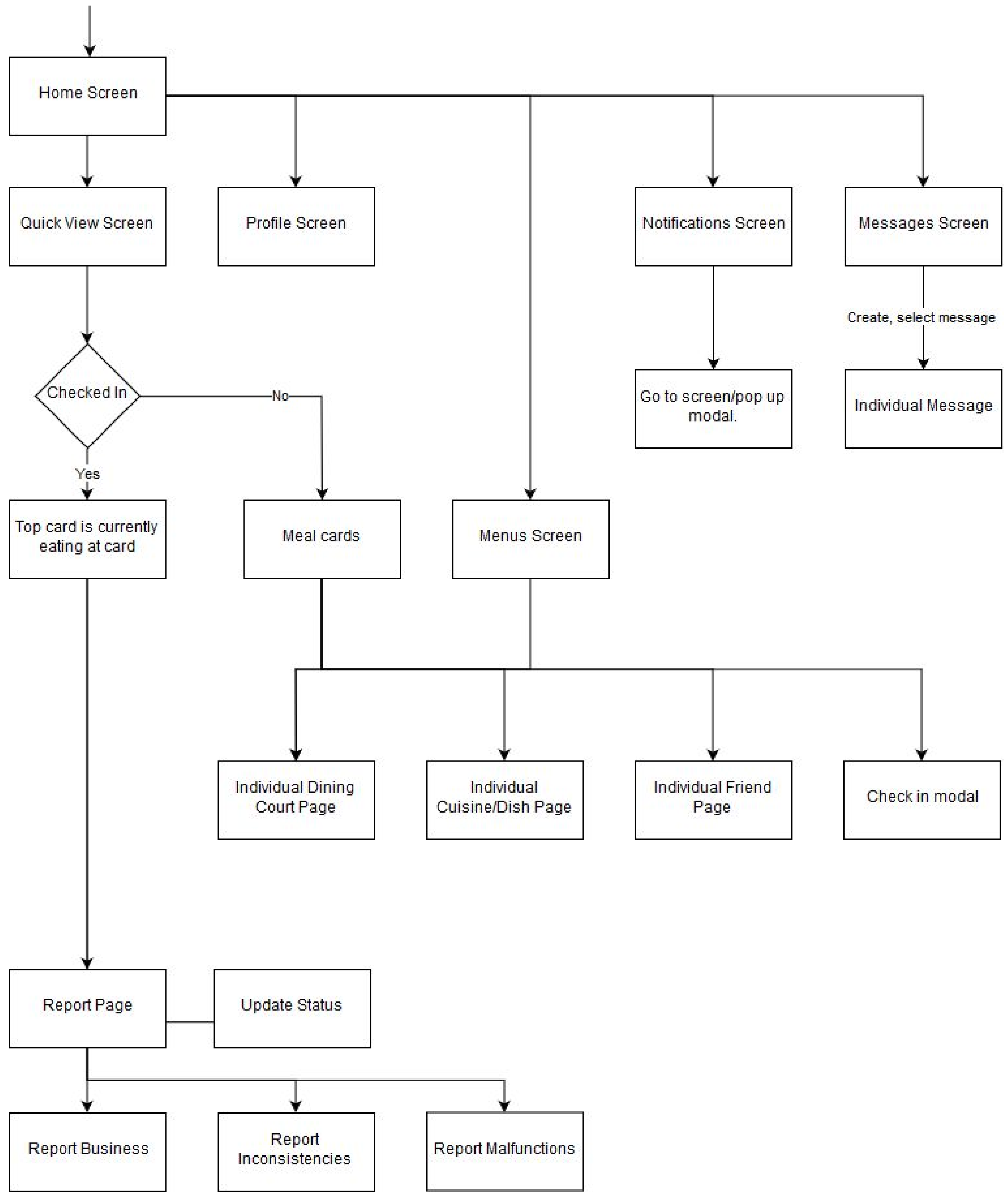
The following are navigation diagrams that describe how the user will interact with the app and navigate through it.

Authentication Navigation Diagram

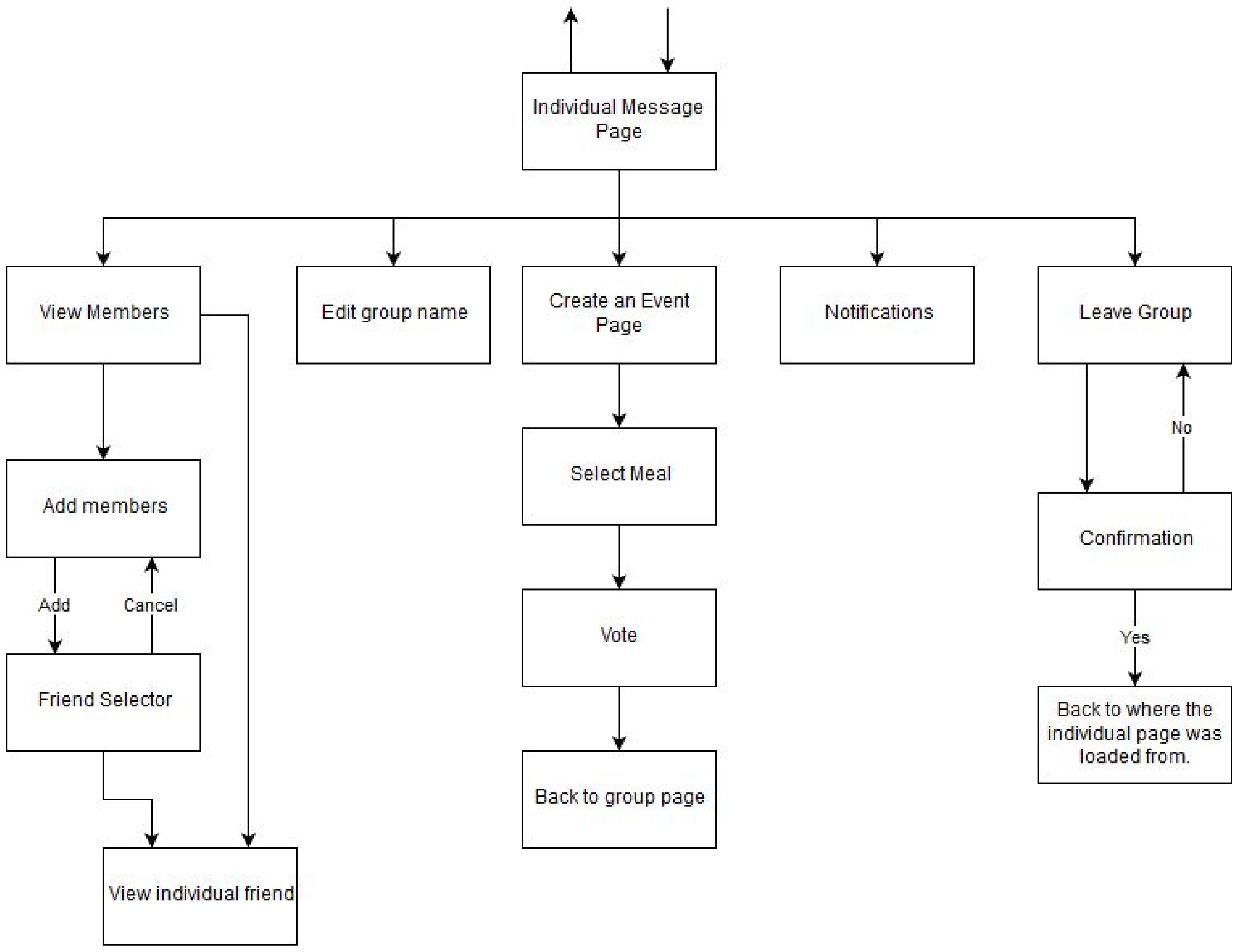


Home Screen Navigation Diagram

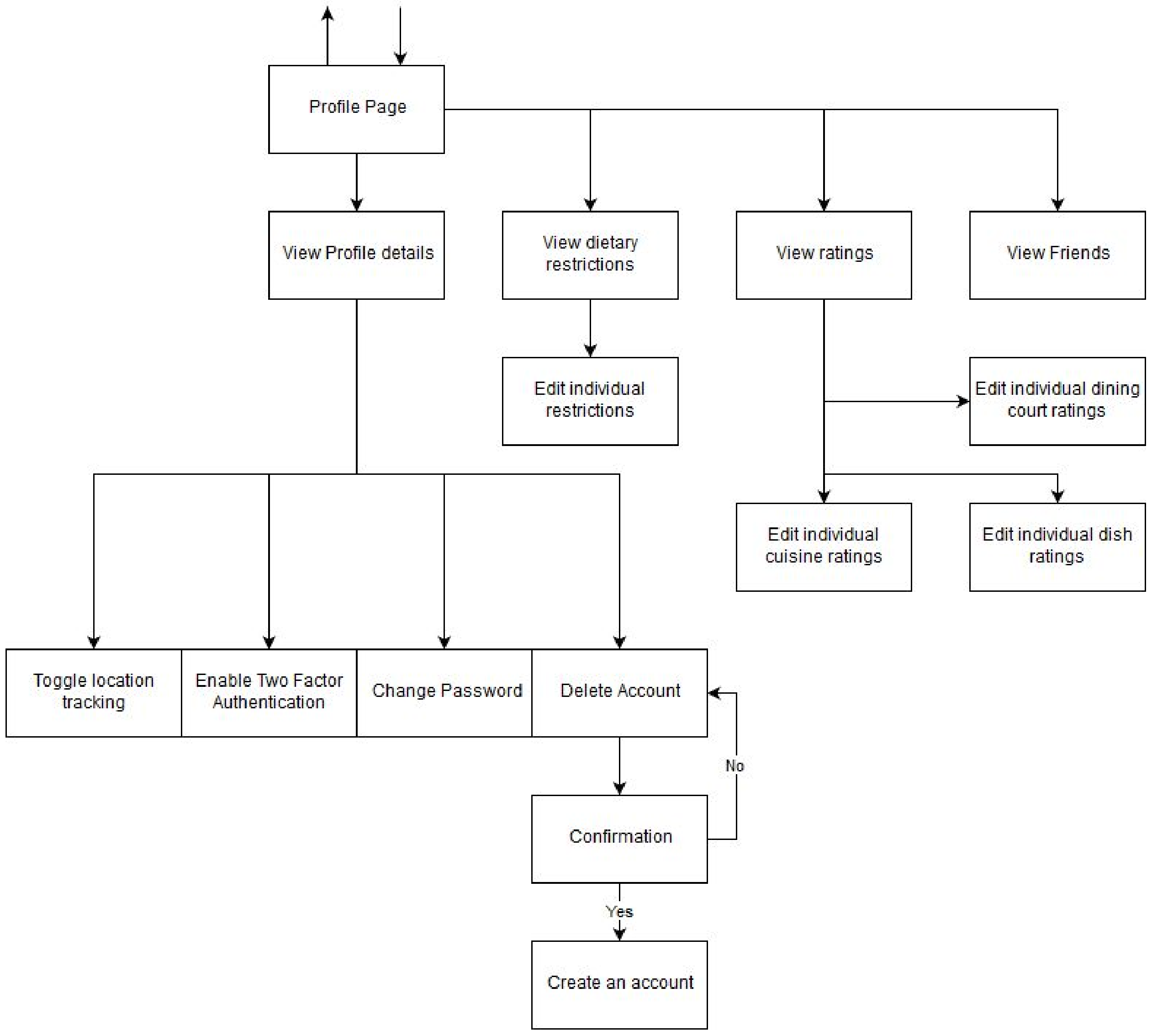
This covers general functionality of the main screens of the app.



Individual Message Navigation Diagram



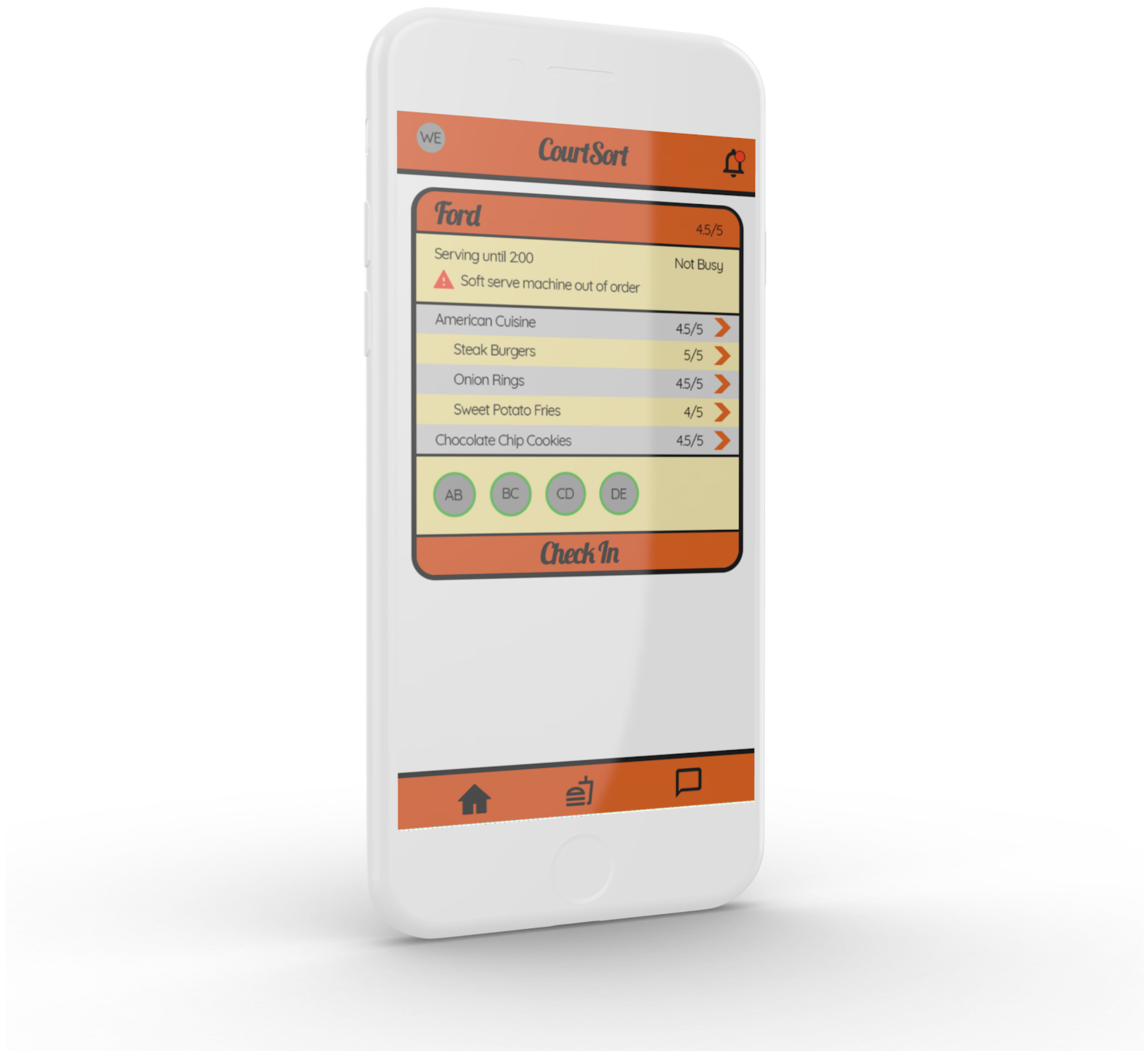
Profile Page Navigation Diagram


Profile Page
View Profile details
View dietary restrictions
View ratings
View Friends
Edit individual restrictions
Edit individual dining court ratings
Edit individual cuisine ratings
Edit individual dish ratings
Toggle location tracking
Enable Two Factor Authentication
Change Password
Delete Account
No
Confirmation
Yes
Create an account

User Interface Mockups

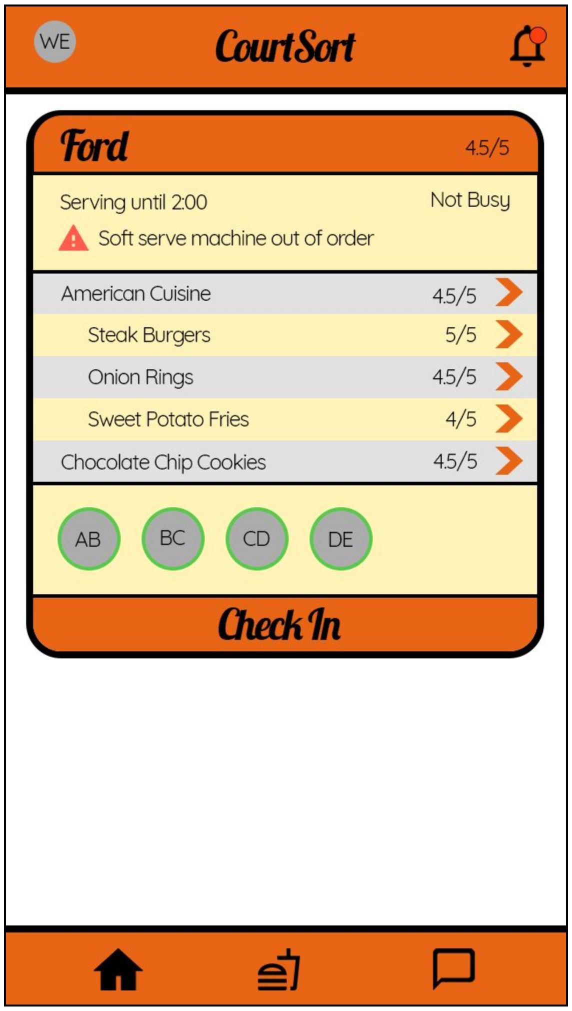
See https://invis.io/RGQ9XIKK3ZS for example user interactions. To use, double click on the device to see connected buttons.

Here is an example view of what it would look like on a phone. The home screen is currently loaded.



Home Screen

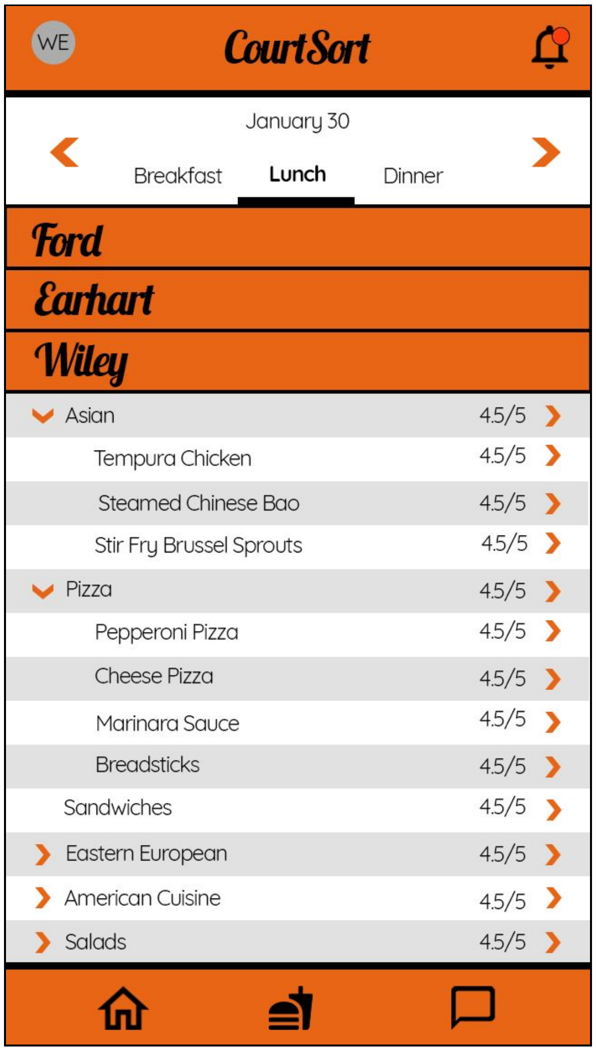
This is the first screen that is loaded. This is where quick information is loaded and is communicated to the user. It is essentially the user dashboard.



American Cuisine	4.5/5
Steak Burgers	5/5
Onion Rings	4.5/5
Sweet Potato Fries	4/5
Chocolate Chip Cookies	4.5/5

Menus Screen

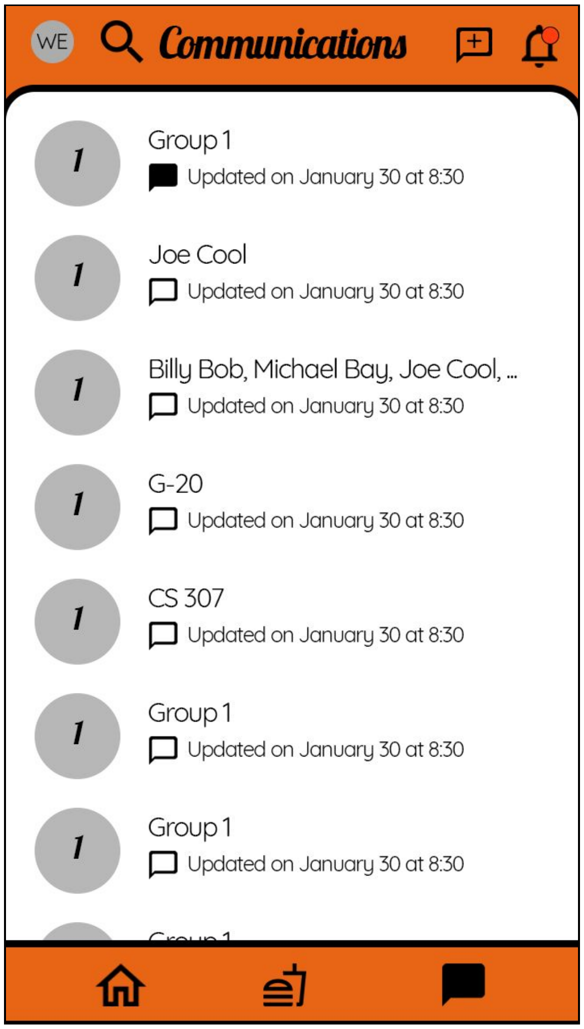
If someone wants to see everything being served at a dining court and at specifics times, this is for them. The date can be changed with the side buttons and the meal with the tabs. Each dining court and item is clickable for more details. Each dining court cuisine can collapse onto itself for fast analysis.



Asian	4.5/5
Tempura Chicken	4.5/5
Steamed Chinese Bao	4.5/5
Stir Fry Brussel Sprouts	4.5/5
Pizza	4.5/5
Pepperoni Pizza	4.5/5
Cheese Pizza	4.5/5
Marinara Sauce	4.5/5
Breadsticks	4.5/5
Sandwiches	4.5/5
Eastern European	4.5/5
American Cuisine	4.5/5
Salads	4.5/5

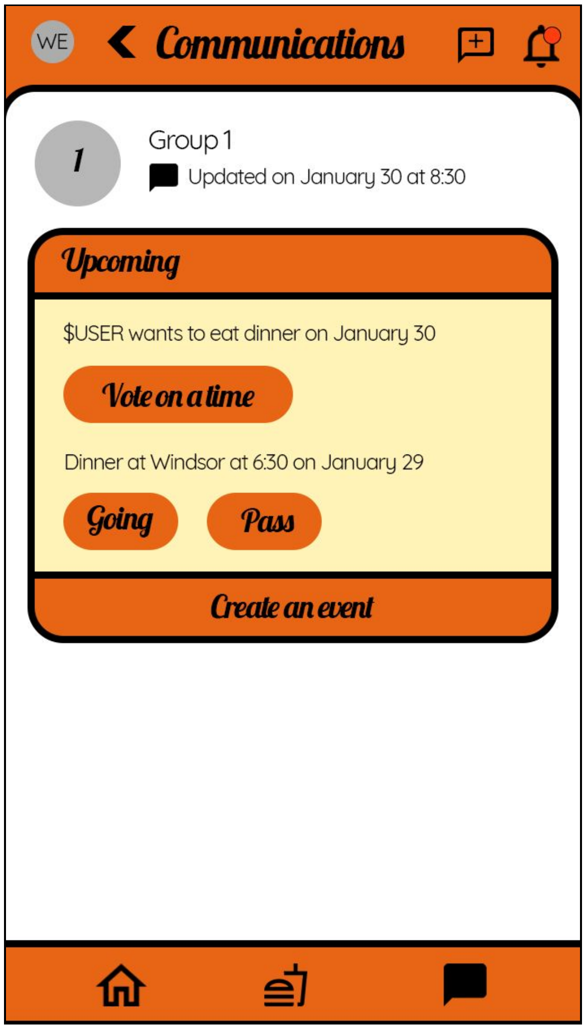
Communications Screen

This is the list of all communications. It contains both group and individual messages. It is a searchable list in last updated order.


WE
Communications
1
Group 1
Updated on January 30 at 8:30
Joe Cool
Updated on January 30 at 8:30
Billy Bob, Michael Bay, Joe Cool, ...
Updated on January 30 at 8:30
G-20
Updated on January 30 at 8:30
CS 307
Updated on January 30 at 8:30
Group 1
Updated on January 30 at 8:30
Group 1
Updated on January 30 at 8:30

Individual Messages Screen

This is where groups and individuals perform communication actions such as creating an event, voting on a time, and responding to an event.



Group 1

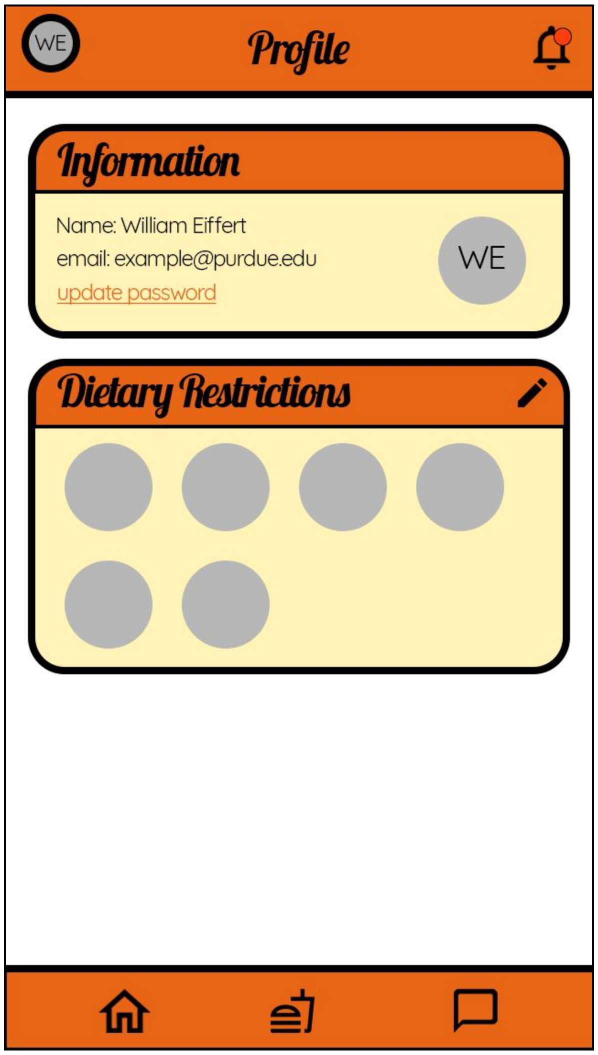
Updated on January 30 at 8:30

$USER wants to eat dinner on January 30

Dinner at Windsor at 6:30 on January 29

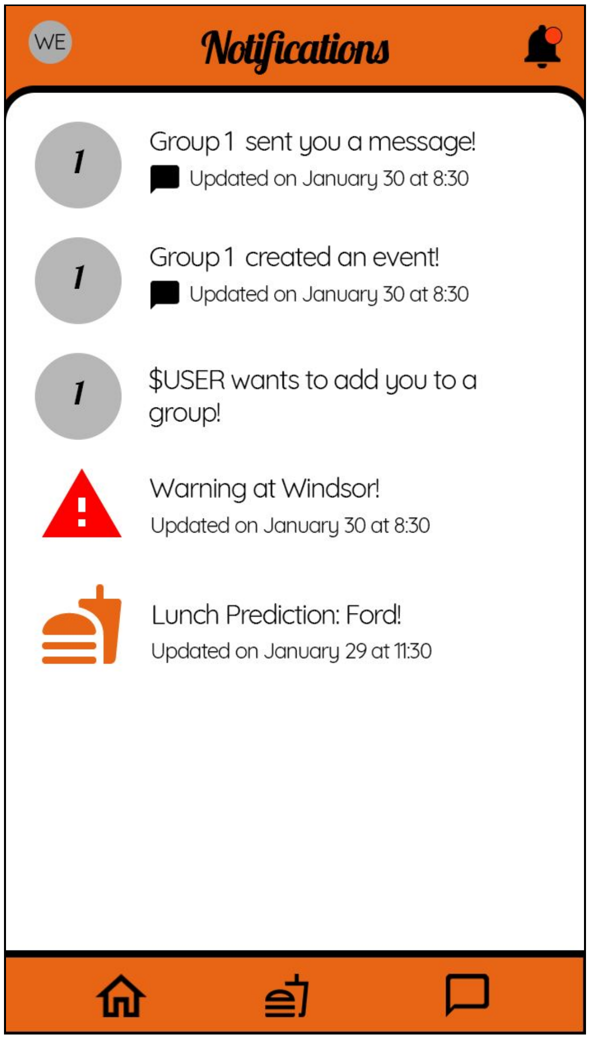
Profile Screen

Here is where users can update their information, dietary restrictions, preferences, and manage who their friends are.



Notifications Screen

This is a quick reference page for warnings, messages, event notifications, and any other notification style interactions. Each element will be clickable and take the user to the related page, or pull up a modal with more information.



Group 1 sent you a message!
Updated on January 30 at 8:30

Group 1 created an event!
Updated on January 30 at 8:30

$USER wants to add you to a group!

Warning at Windsor!
Updated on January 30 at 8:30

Lunch Prediction: Ford!
Updated on January 29 at 11:30

Icons used are Google's Material Design Icons. See https://material.io/ for more information.