Daniel Lerner, Izzy Gomez, Brian Saavedra, Sara Stiklickas
dibs

Design

Motivation

The system we are building connects event hosts and guests in a way that streamlines the experience of ordering drinks. The app has a few key purposes:

- Let event hosts know what drinks are popular among their guests
  - Right now, there isn't a good way for a host to get information about guests' preferences. A host can hold a much more successful party if he or she knows what the guests will want to drink.
- Prevent bars or serving areas from getting too crowded
  - When guests have to wait near the serving area for their drinks to be prepared, it can get very crowded and uncomfortable. This can cause a fire hazard, which is a danger that should be prevented. Additionally, guests generally don't like having to wait a long time for their drinks.
- Control drink distribution
  - Events often have the problem of distributing a limited supply of drinks to a large number of guests, so hosts might want a way to impose a drink limit and distribute the drinks more evenly.
- Make sure the right people get drinks
  - Some events might be exclusive, only supplying drinks for invited guests. Hosts of such events need a way to ensure that unwanted guests don't show up and order drinks when they shouldn't.

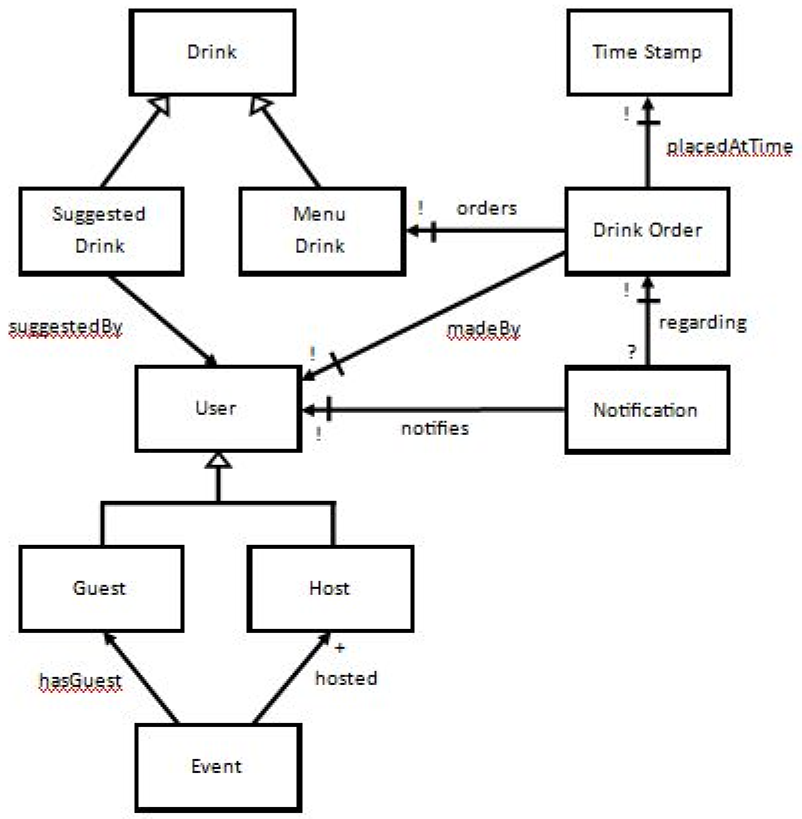
Concepts

1. Drink order
   Purpose:
      To allow guests to place a drink order.
   Operational Principle:
      A guest at an event places an order, from the drinks that are currently in stock. The order is placed on the queue that the host can see, and when the drink is made, the guest receives a notification, saying that the drink is ready.
2. Order Queue
   Purpose:
      To allow the host to see which drinks he has to make next, in the order in which they were requested.
   Operational Principle:
      A host looks at the request at the top of the queue, and makes the requested drink. When the drink is ready, he checks the drink off, and the user is informed.

When the user gets the drink, the host verifies that this is the person that actually ordered the drink by looking at a picture (provided by Facebook), and then checks the drink as being received, so it is completely removed from the queue.

3. Suggestions
   Purpose:
      To allow a guest to suggest a drink to have at an event before the event occurs.
   Operational Principle:
      A guest can make a limited amount of suggestions for drinks he wants to see at an event. The host looks at this information, and decides which drinks he will buy to satisfy the needs of the group of people who will be attending the event.
4. Menu
   Purpose:
      To show guests which drinks are available for them to order, along with how many of each kind remain allotted to them.
   Operational Principle:
      When a guest wants to place an order, he must select a drink from the menu, which only shows him drinks that are in stock at the event and of which he hasn't reached his limit.

Data Model



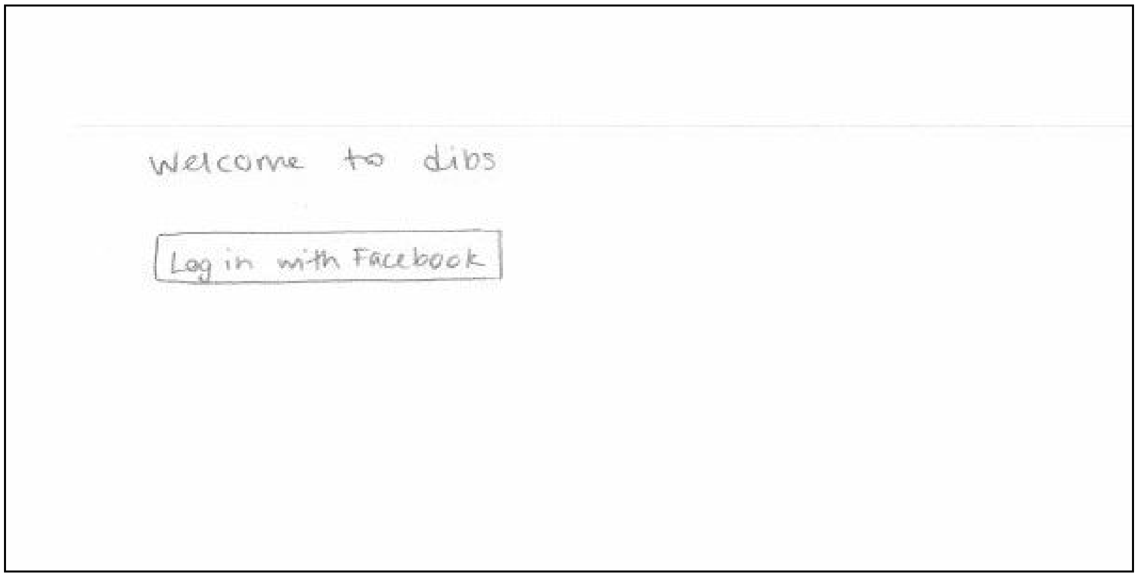
Security Concerns

- Code injection is possible when users submit drink requests before an event
  - Can be prevented through sanitization of inputs.
- Code injection could occur through Facebook API if it has security vulnerabilities
  - (very) unlikely, but possible and still a concern
- Depending on how we use the Facebook API through our application, it is possible to have CSRF attack if we make requests to FB servers through a the client.
  - For example, a request embedded in a simple <img> tag on a malicious website while our app is open can (maliciously) use the client-side of the app make use of the FB API
  - Solution to this would be to only interact with the API only on the server
  - This, however, will also necessitate measures against code injection to our server codebase through the client-server interface (i.e. sanitization as mentioned earlier)
- Threat model: Thus, our threat model is the set of the possible security flaws already mentioned. These include unsafe API usage, such as sending requests through the

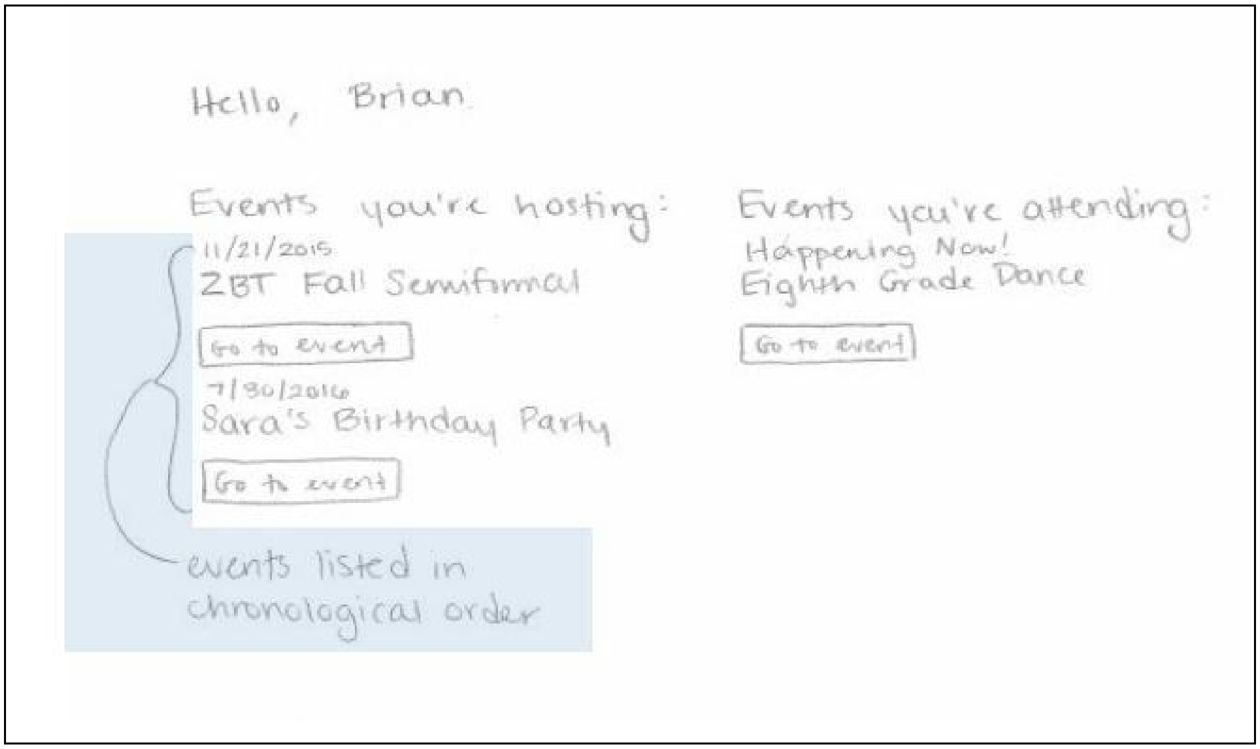
client, or malicious code injections, such as during any user input event (registration, drink suggestion, drink request, …).

User Interface

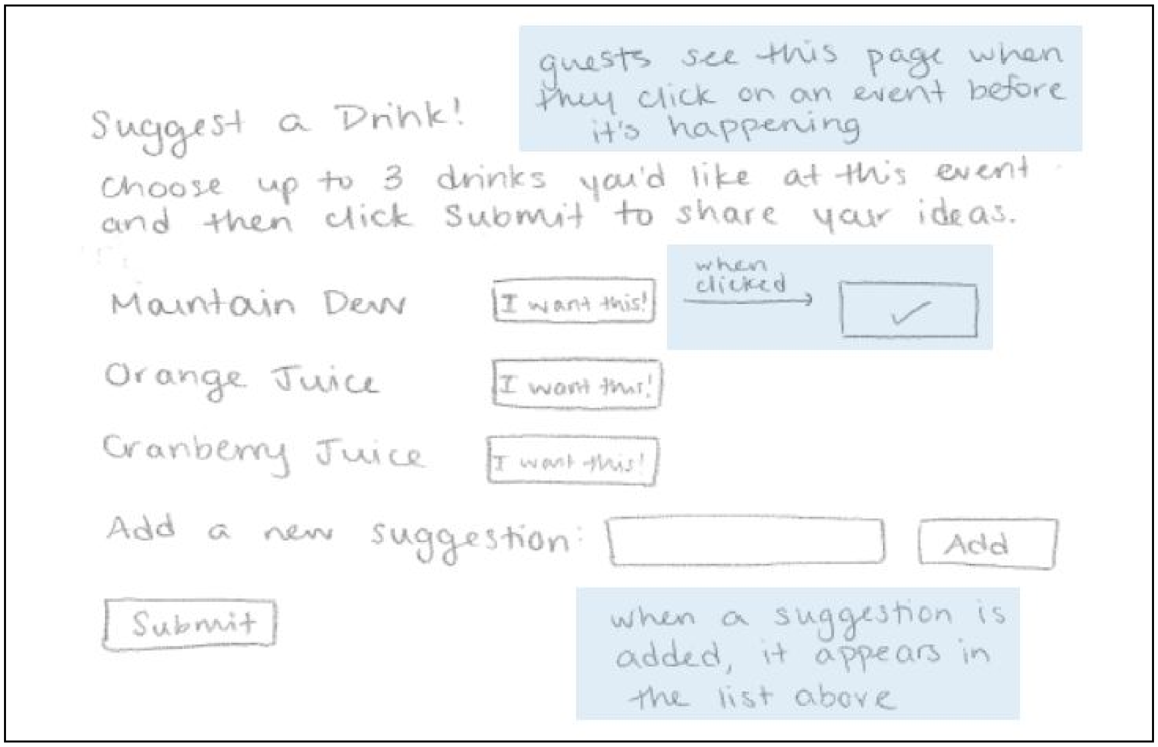
1) Welcome Screen



2) Event Page



3) User Suggestion Page


Suggest a Drink!

guests see this page when they click on an event before it's happening

Choose up to 3 drinks you'd like at this event and then click Submit to share your ideas.

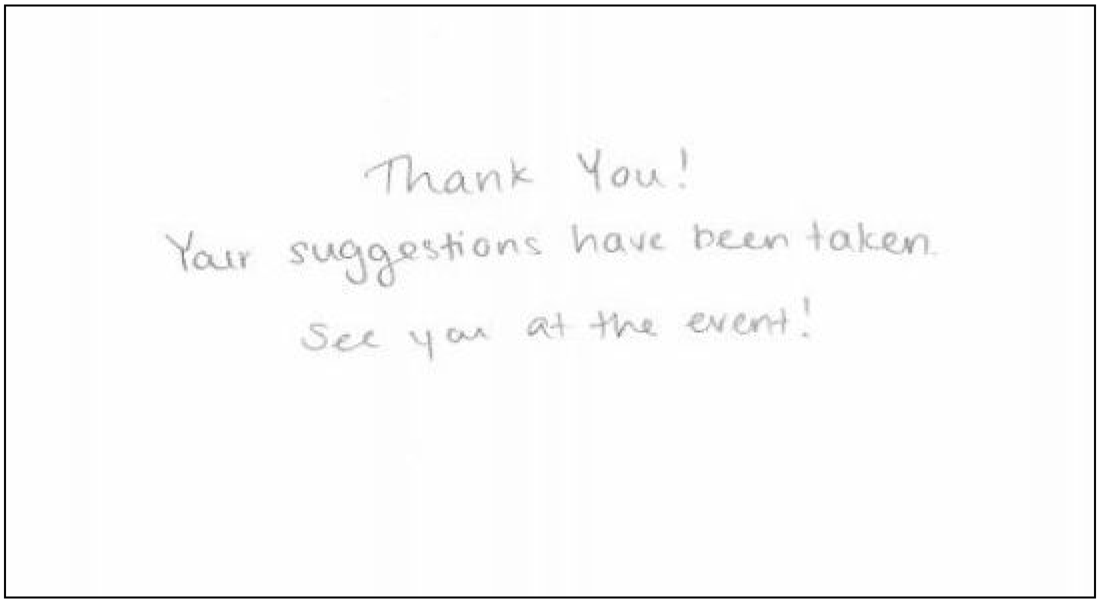
Mountain Dew	I want this!
Orange Juice	I want this!
Cranberry Juice	I want this!

Add a new suggestion: Add

Submit

when a suggestion is added, it appears in the list above

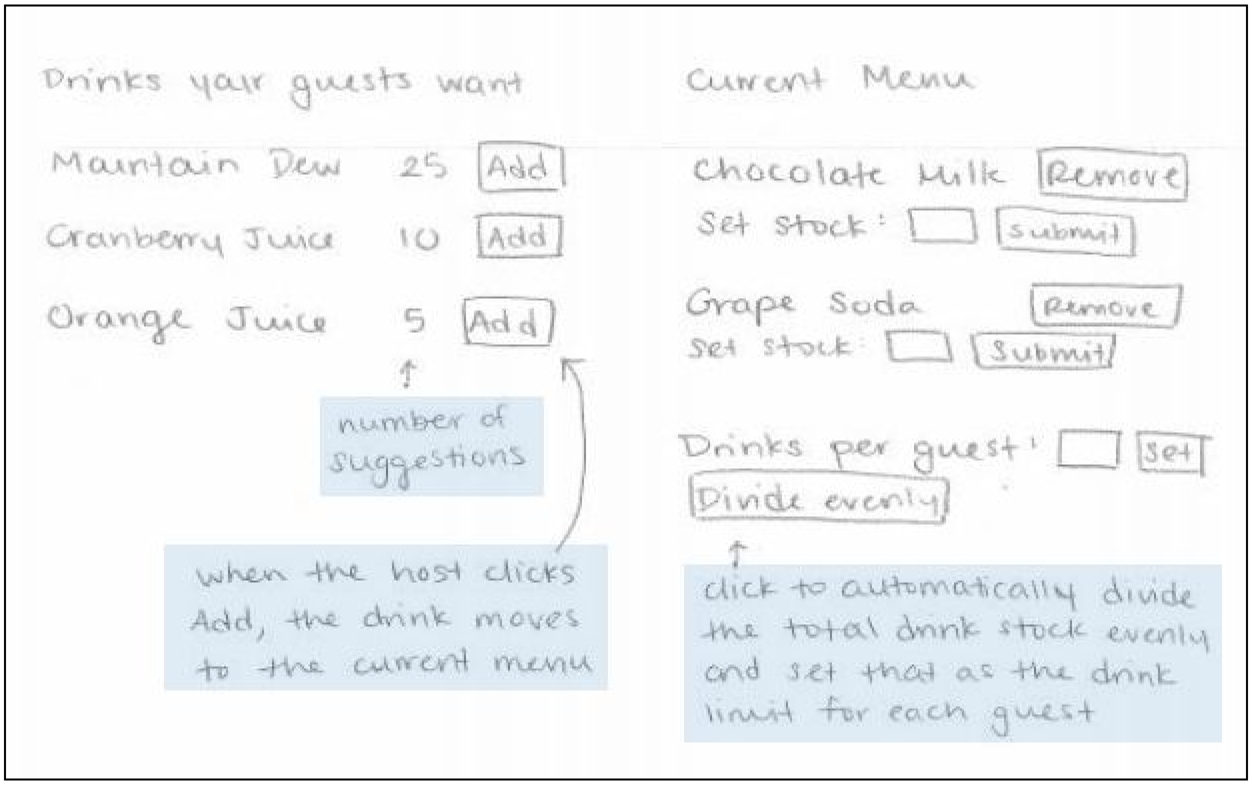
4) Suggestion Thank You Page


Thank You!

Your suggestions have been taken.

See you at the event!

5) Host Suggestion Page

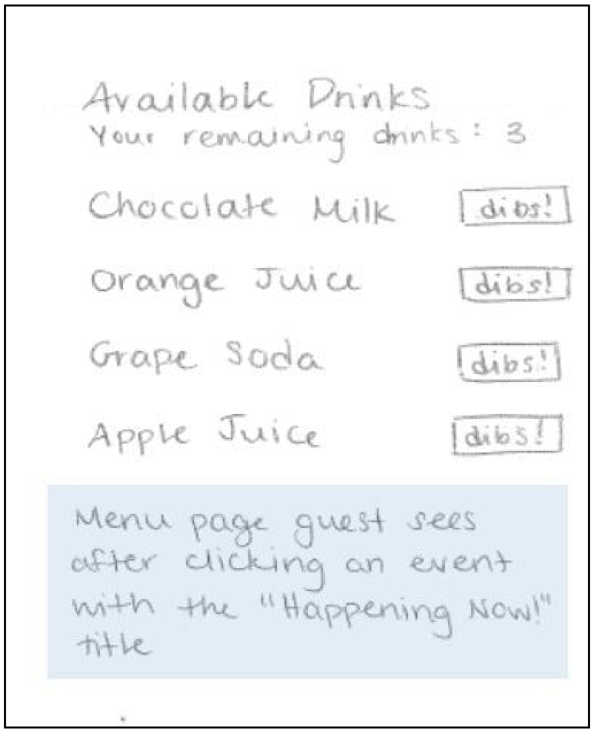


number of suggestions

when the host clicks Add, the drink moves to the current menu

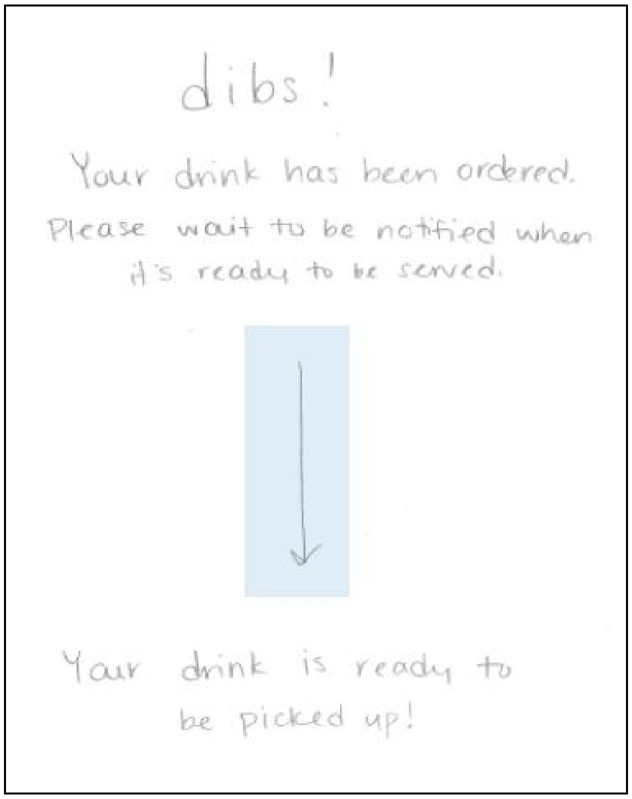
click to automatically divide the total drink stock evenly and set that as the drink limit for each guest

6) User Dibs Page

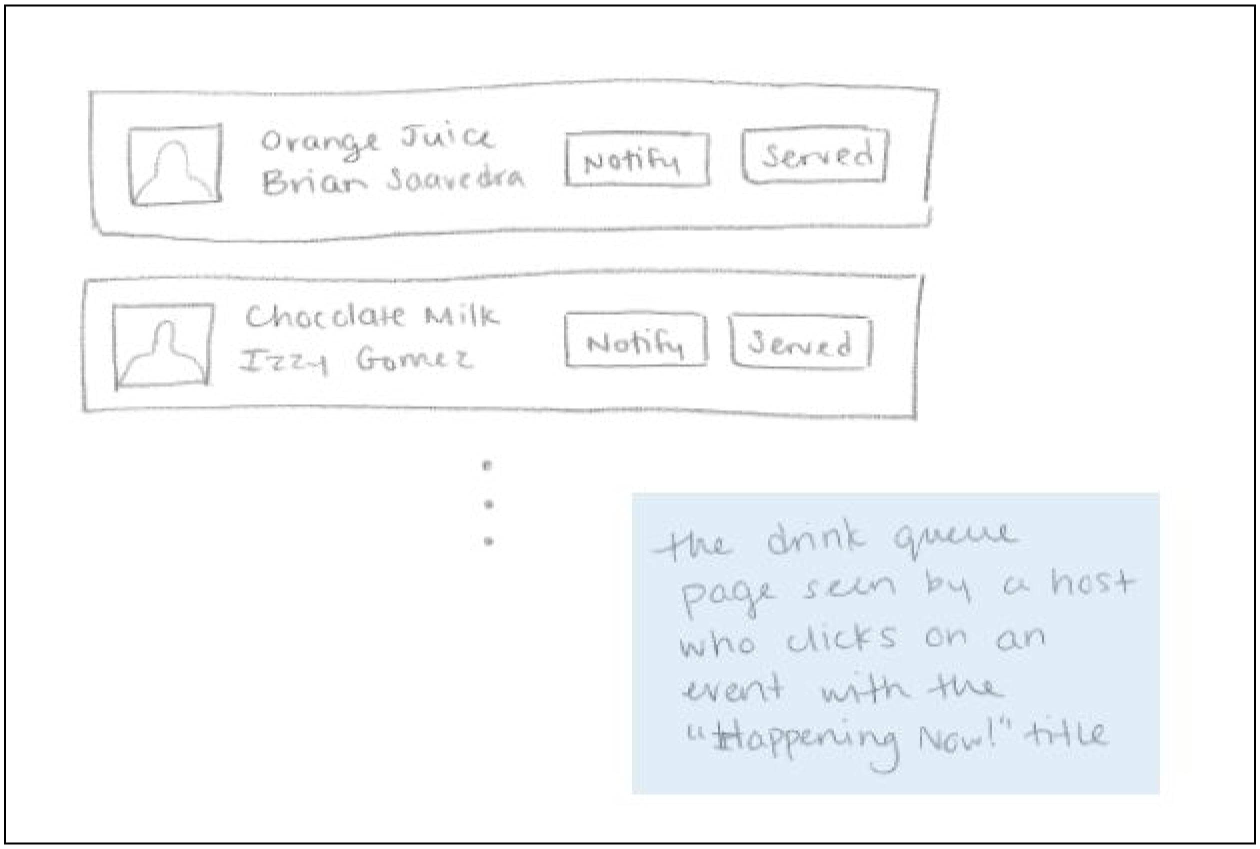


Menu page guest sees after clicking an event with the "Happening Now!" title

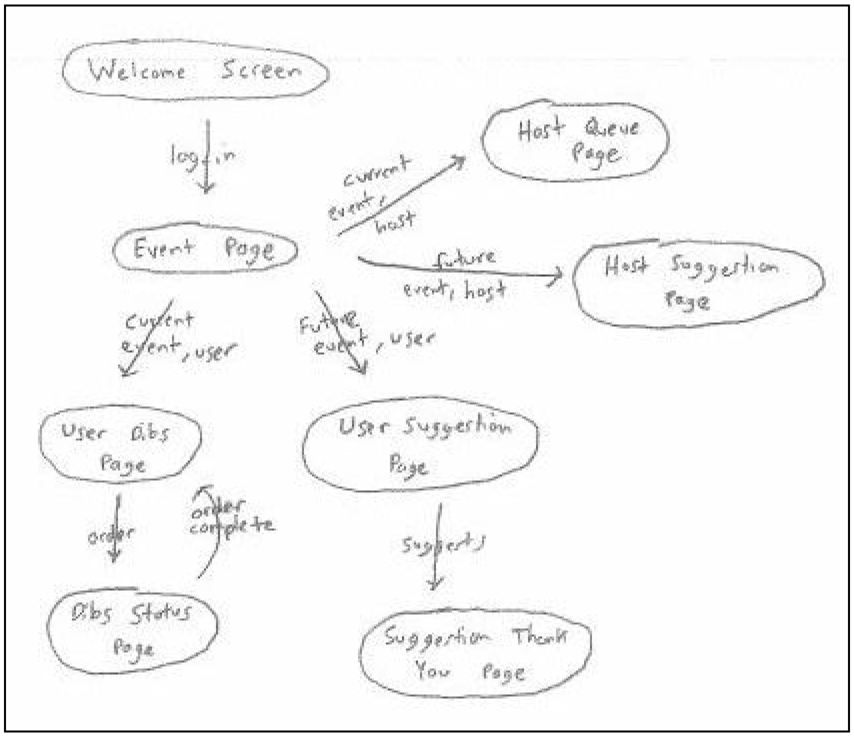
7) Dibs Status Page


dibs!
Your drink has been ordered.
Please wait to be notified when it's ready to be served.
Your drink is ready to be picked up!

8) Host Queue Page


Orange Juice
Brian Saavedra
Notify
Served
Chocolate Milk
Izzy Gomez
Notify
Served
the drink queue page seen by a host who clicks on an event with the "Happening Now!" title

UI Transition Diagram



Design Challenges

List of problems to resolve in concepts, data model or user interface For each problem: options available, evaluation, which chosen Data design choices and their justifications

Data Model:

- The current data model doesn't clearly define the relationship between the different users of our app (hosts and guests) and their interaction with the sets in our data model. For instance, the relations expressed in the data model show the associative relationship between sets such as the connection between a Drink Order and the specific Menu Drink that was ordered, but doesn't express the app's view-rights of how only the host may see this order and interact with it. This problem is an inconvenience when trying to get a clear implementation guide since different user functionality is not expressed. Our mitigation for this problem is to continually revisit our data model as we're starting our app implementation to update it and avoid design pitfalls early in our work.

User Interface:

- *Notification system*: because our app is web-based, it's hard to implement a real-time notification system. Users will presumably access our app through their phone's browser while at an event, which unfortunately doesn't support push notifications. The options available are to have a simple status page which is continuously refreshed to see if it's been updated with some sort of "drink is ready" message, or to implement an SMS notification system into our application. We've decided to go with the former option as it simplifies implementation and doesn’t make assumptions about how our user is using our application (it might be the case that a user brought their laptop around to the event).