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. Before an event, guests can use it to suggest drinks, giving the host ideas about what to serve. During an event, guests can order drinks that are available, letting the hosts know what to make in real time. 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. Dibs
   Purpose:

   To allow guests to place a drink order.

   Operational Principle:

   A guest at an event calls dibs on a drink that is currently in stock.

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.

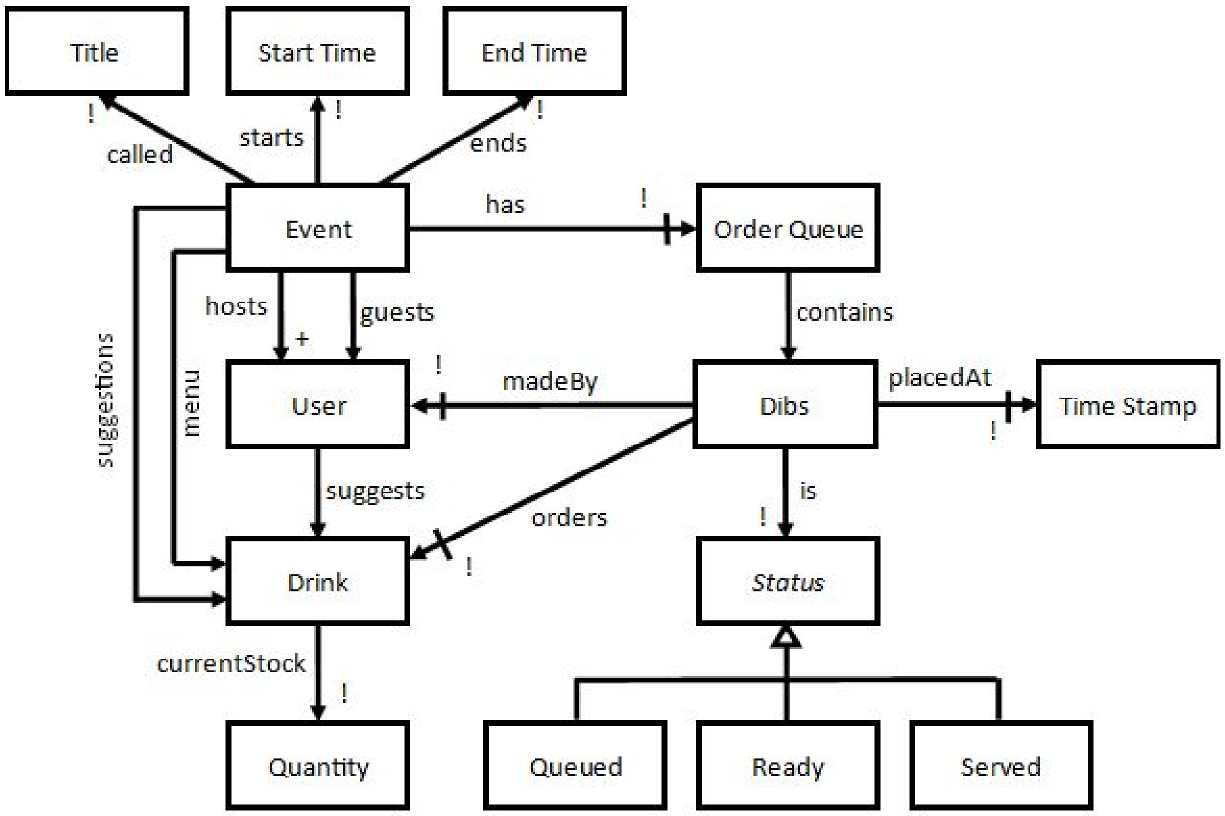
   Operational Principle:

A host/server looks at the request at the top of the queue, and makes the requested drink.

3. Suggestions
   Purpose:
   To allow a guest to make a limited amount of suggestions for drinks to have at an event before the event occurs.
   Operational Principle:
   A guest makes a suggestion for a drink he wants to see at an event and the host can use these suggestions when deciding what to serve.

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:
   A guests selects a drink from the menu and the stock of that drink is decreased by one.

Data Model



Textual Constraints:

A guest can only make a drink order for a drink on the menu at that event.

Hosts cannot be guests and guests cannot be hosts for the same event.

A drink order cannot be placed for a drink whose current stock is empty.

Clarifications:

Calling "dibs" is the app's name for placing a drink order.

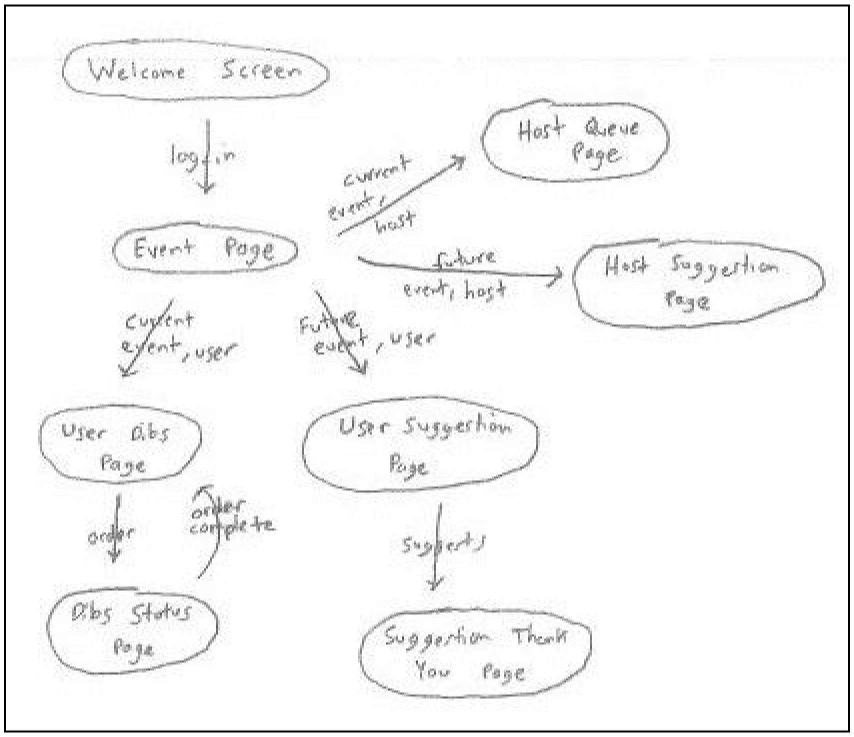
Security Concerns

- 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, …).

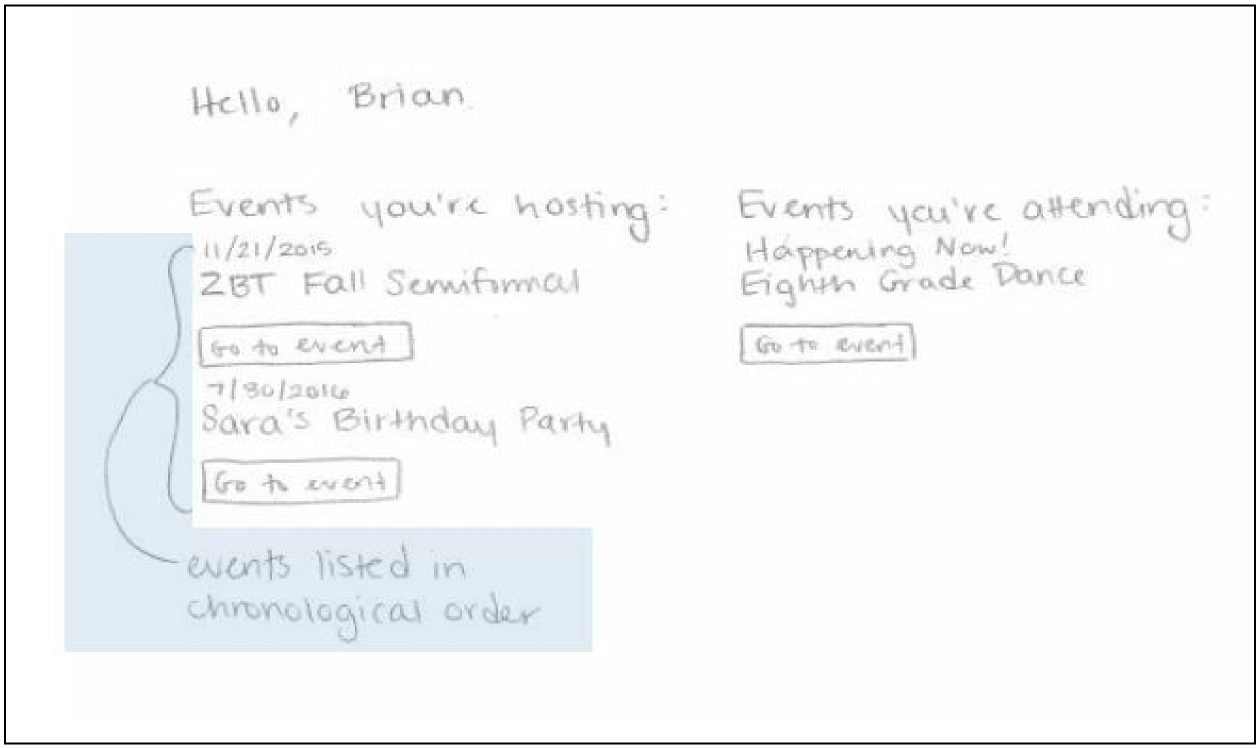
User Interface

UI Transition Diagram



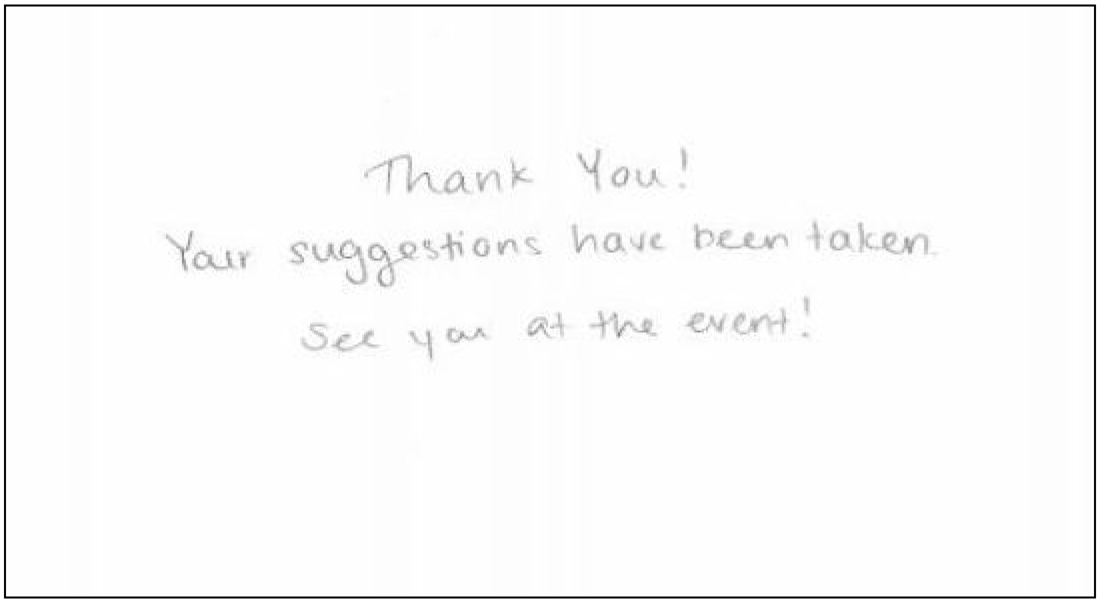
1) Welcome Screen

2) Event Page



3) User Suggestion Page

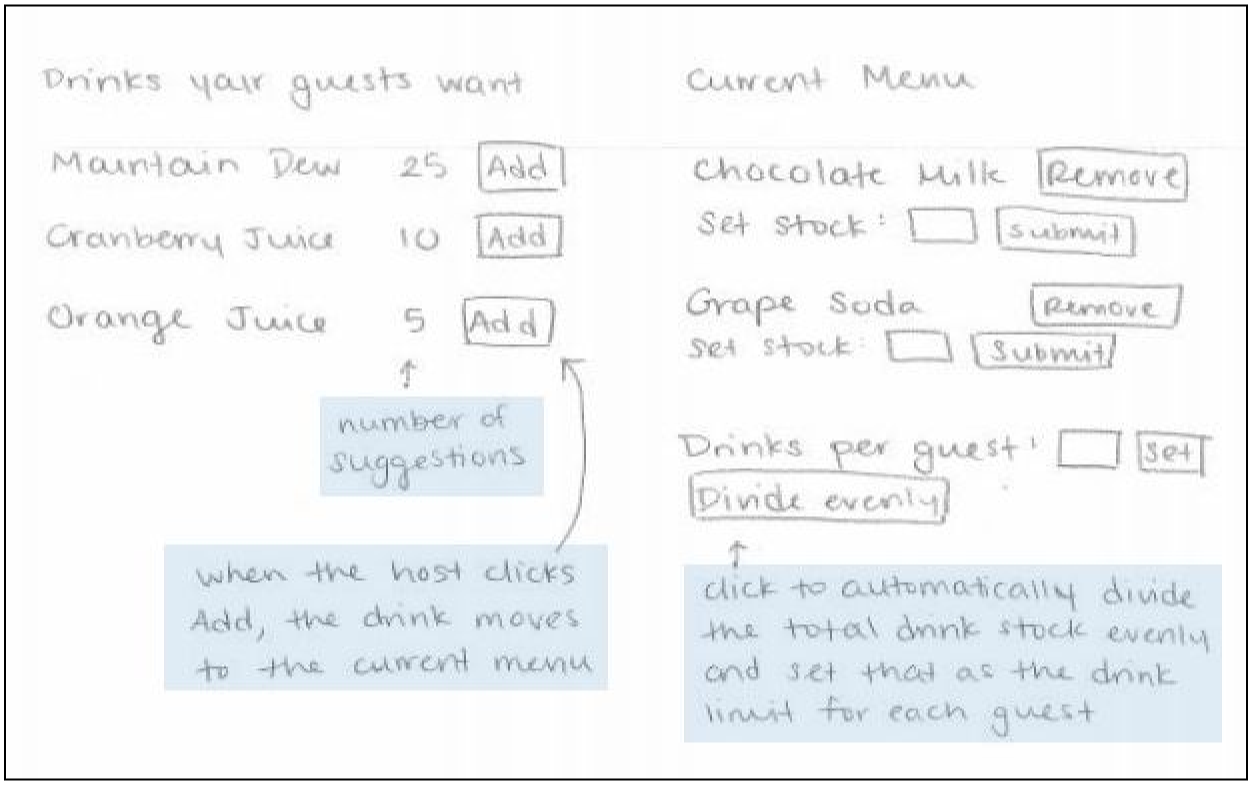
4) Suggestion Thank You Page


Thank You!

Your suggestions have been taken

See you at the event!

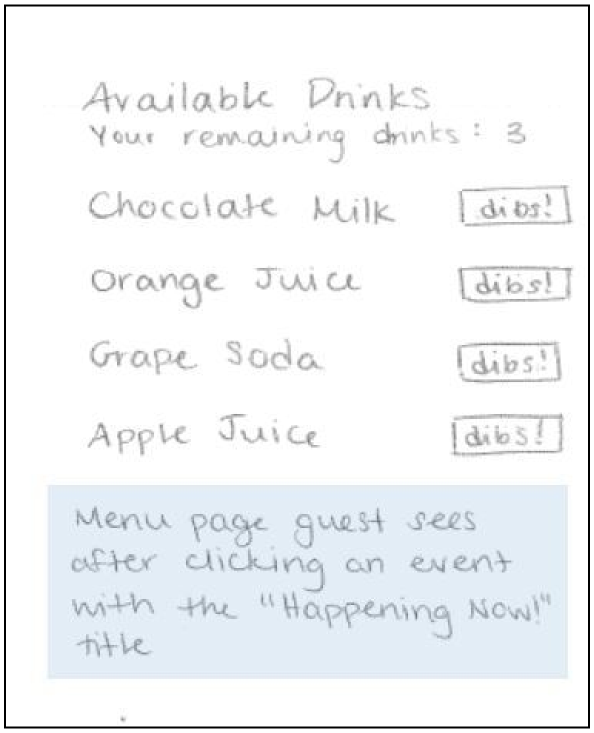
5) Host Suggestion Page



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

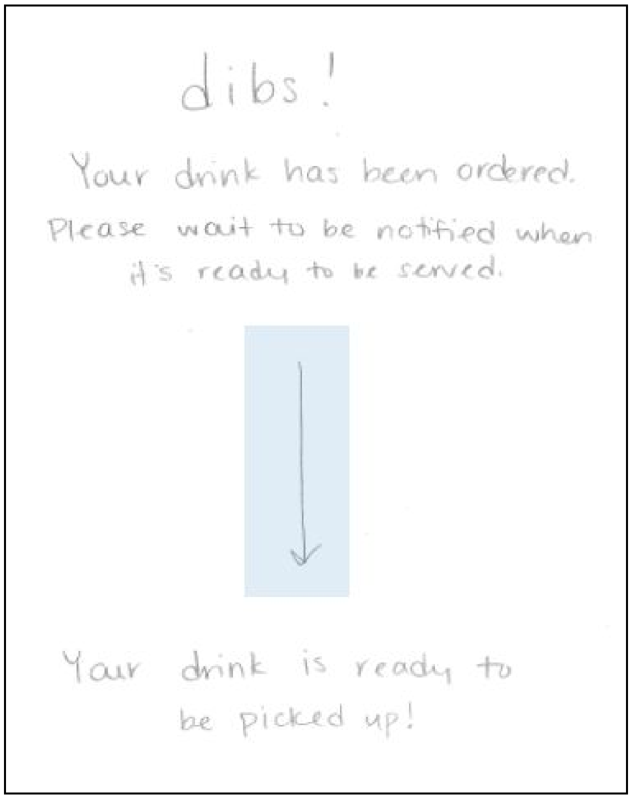
click to automatically divide the total drink stock evenly and set that as the drink limit for each guest

6) User Dibs Page

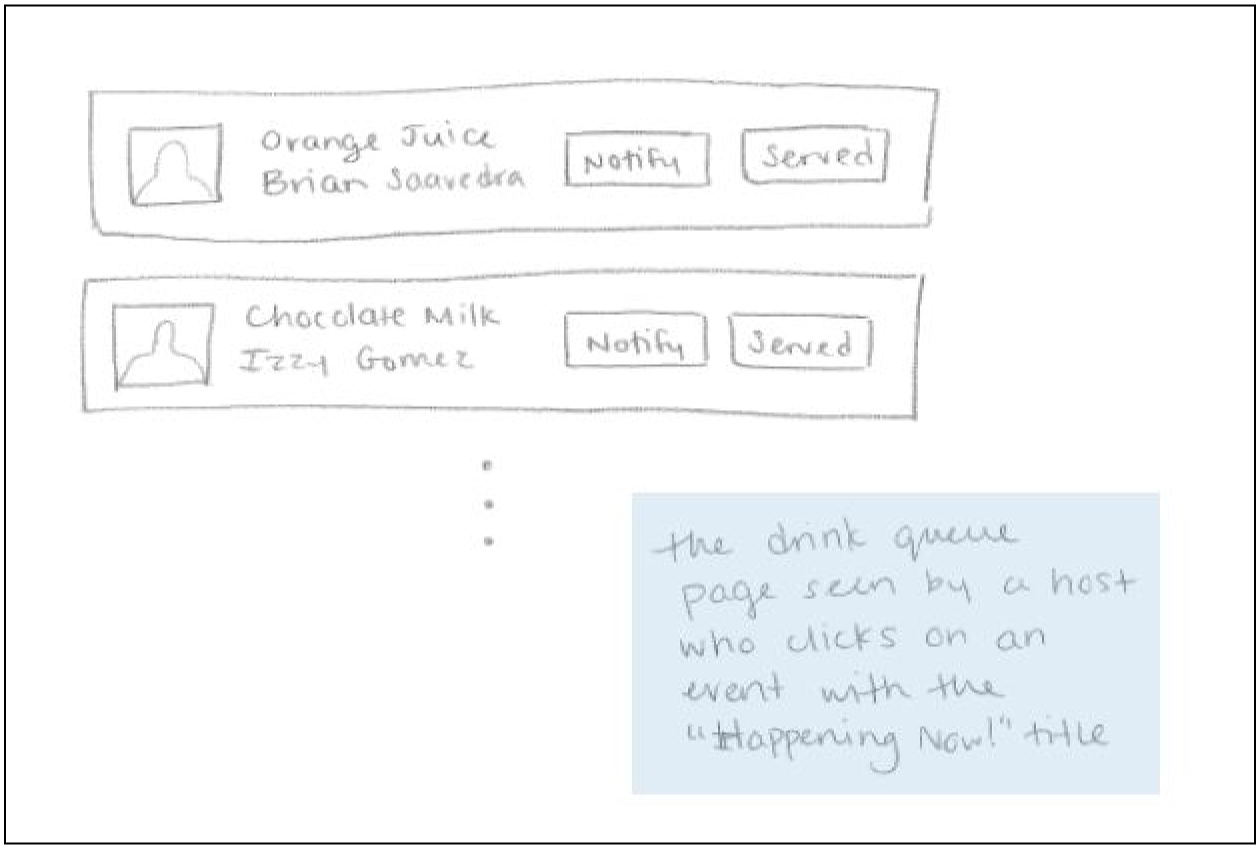


Menu page guest sees after clicking an event with the "Happening Now!" title

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

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:

- *Event:* In our original Data Model, an Event was out of focus and lacked relations to many crucial sets such as the Order Queue and Drink Suggestions. Our options consisted of leaving Event out of focus in our data model or revising our data model to make an Event more centralized. We did not find very many pros for leaving Event out of focus and for this reason, decided to revise our data model with Event being a main focus, having relations to Drink, the Order Queue for that event, and other key components.

- *User*: Previously, our Data Model did not allow guests to be hosts at another event and a host at one event to be a guest of another. This was a major flaw in that it did not represent our true intention for users. In order to resolve this issue, we considered making Guest and Host relations instead of subsets of User. We chose to create new relations for Guest and Host because adding a textual constraint would not resolve the design flaw.

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 latter option as it doesn’t require the user continually checking the web page to see if a drink is ready.