Full Design Revision
6.170 Software Studio
Capolino, Moustaklem, Karimi, Vostatek
11/29/2018

Revision Summary:

Anonymity
Given our late meeting with Suzy Nelson on Dec. 7th, we decided to remove the anonymity part from the design. Hence students will not be able to request an anonymous meal. However, the design as is is ready for change in case the meeting is successful (everything that needs to be added will be simply implementation issues).

OpenID Connect
Our goal for the final application submission will be using OpenID Connect Api to check for the validity of the kerberos, and then use regular email authentication to validate that the registering user actually controls that account. This will allow us to circumvent the issue of multiple sign-ins using MIT mailing lists while also validating users!

Our ***stretch*** goal for the final application submission, and for the finished application regardless of 6.170, will be using OpenID Connect Api to log students in with their kerberos or certificates. With this, the users will be redirected to the Open ID page where it will give them the option of logging in with their certificate or with their kerberos and password. The reason for this is two fold: firstly, we want to provide users with a more convenient way of logging in. Secondly, we are doing this to avoid a security flaw of allowing users to log into the app using an MIT mailing list. If we just used MIT emails to authenticate, a non-MIT student could be placed on a mailing list and therefore access the application.

Canceling Meal and Request
We will add actions for canceling a meal once two people have been match, and a request once it is still pending.

Database cleanup
As it stands now, the requests table in our database can contain expired requests. That is, the date of the request can be in the past relative to the present date. Our new design will take this into consideration and periodically (every night) delete all requests which are expired. If the requests are fulfilled, they are similarly deleted from the table and an entry is created in the 'meals' table. We decided against storing fulfilled requests because it creates storage overhead, reduces speed, and complicates the queries for our matching algorithm. Finally, we will also be adding a cap of 5 outstanding (unfulfilled) requests per user. This way a user can not spam our app with requests and matching is more naturally evenly distributed.

Below is the Full Design with changes highlighted in Bold.

Overview

Description

I Have This Food is a platform that matches students who have excess dining hall meal swipes and students who would like to receive these meal swipes. The system will allow for **one** ~~**two**~~ mode~~**s**~~ of exchange**:**~~**, an anonymous online transfer of the meal swipe from owner to receiver, and**~~ a meet-up option where the two students are matched at a dining hall for a certain time frame in which one can swipe the other in.

Key Purposes

1) *I Have This Food* mitigates the issue of food insecurity on MIT's campus, which a recent study found to be prevalent. This purpose addresses the following problems:
    a) Student lack of food security
        i) Many students on campus are unable to find food when they need it
        ii) *IHTF* will allow students in this position to find meals.
    ~~**b) Stigma associated with seeking help**~~
        ~~**i) It can be sometimes shaming for a person in a position of need to ask for a meal**~~
        ~~**ii) IHTF will provide an anonymous mechanism in which students can digitally transfer a specified number of swipes to whoever requires them.**~~
2) *I Have This Food* helps students utilize their meal plans more efficiently. This purpose addresses the problems listed below:
    a) Meal plans are often underutilized
        i) Many students find themselves with extra swipes, whether guest swipes or block plan meal swipes, at the end of the semester and often do not use them
        ii) *IHTF* allows students on both types of meal plans to increase utilization of their meal plans by facilitating sharing of meal swipes
    b) Excessive food waste
        i) Unsurprisingly, there is food waste on campus dining halls, however, this is especially troubling given that there are students on campus in need of that food that is being discarded at the end of the day
        ii) *IHTF* makes it such that less food will be wasted as the utilization of meal swipes increases due to the easy sharing process.

Deficiencies in Current Solutions

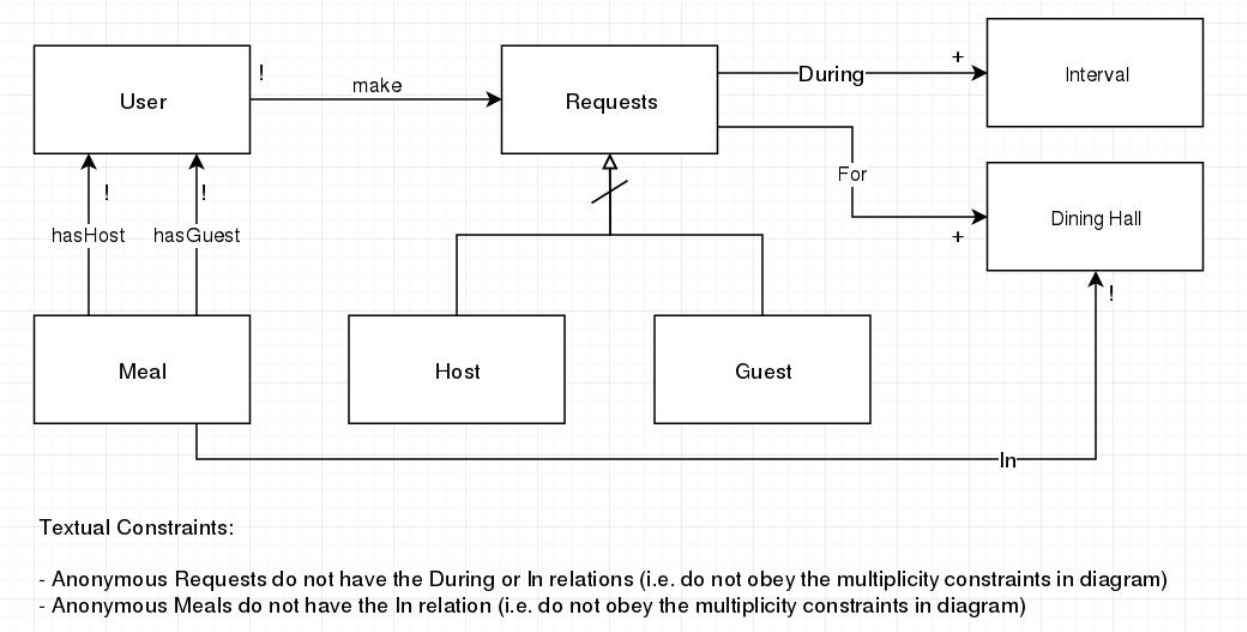
1) SwipeShare is the only current existing solution that claims to address this issue. The deficiencies with this current system are:
    a) Limited number of swipe transfers
        i) Currently a student is only able to give up to 6 meals per semester
        ii) More concerning is that students in need are only able to receive up to 3 meals a semester
        iii) This is inadequate as we believe students suffering from food insecurity will need more than 3 meals a semester to feel secure
    b) Outdated un-intuitive interface
        i) The current interface violates many design heuristics, some of which are:
            (1) Fitt's Law: the size of relevant buttons and information presented on the swipeshare page
            (2) Information Scent: it is not clear from the mycard.mit.edu where one can donate a meal, unless they already know about SwipeShare, and then again if they do, the information is presented in a cluttered manner on that page.
    c) BREAKING NEWS: SwipeShare requires getting approval from a Dean
        i) It could be shaming to go through the approval
        ii) It could be too lengthy of a process before securing the 3 available meals

Conceptual Design

Elaborate the concepts in your conceptual sketch into a full conceptual design, giving for each of the key concepts:

- The name of the concept
- The concept's purpose
- The structure, expressed as an abstract data model
- The behavior, expressed as the actions and their effects on the data model
- The operational principle: a motivating scenario

1. Meal
    a. <u>Purpose</u>
    To mitigate food insecurity among MIT students
    b. <u>Data Model</u>



    c. <u>Behavior</u>
        i. **User actions**
            1. Make_Host_Request(user: User, ~~**anonymity: Bool,**~~ times: Array[Interval], places: Array[Dining Hall])
                a. Req: None, user already confirmed student
                b. Effect: a user host (donating) request is created for the specified times and places, ~~**and will be anonymous if that parameter is true.**~~
            2. Make_Guest_Request(user: User, Array[Interval], places: Array[Dining Hall])
                a. Req: None, user already confirmed student
                b. Effect: a guest user (receiving) request is created for the specified times and places

3. **Cancel_Request(user: User, request: Request):**
    a. **Req: user is owner of request**
    b. **Effect: Request -= request**
4. **Cancel_Meal(user: User, meal: Meal):**
    a. **Req: user to be one of two matched user in meal**
    b. **Effect: Meal -= meal**

ii. **System Actions**

1. Match(hosts: Array[Host Request], guests: Array[Guest Request])
    a. Req: None
    b. Effects: Calls Create_Meal on appropriately matching pairs of guest and host requests based on request parameters and removes them from hosts and guests
2. Create_Meal(host: User, guest: User, location: Dining Hall, time: Time, ~~**anonymity: Bool**~~)
    a. Req:
        ~~**i. If anonymity is true, then there can be no location and no time**~~
        ii. If anonymity is false, then there must be one location and one time
    b. Effects:
        i. Creates a Meal with hasGuest = guest and hasHost = host and In = location and time = time and calls Notify_Match on the meal
            ~~**1. If anonymity = true additionally calls Transfer_Meal**~~
3. Notify_Match(Meal: meal)
    a. Req: None
    b. Effects:
        ~~**i. If Meal.anonymity = true: notifies guest user of the newly available meal and notifies the host user of the match without revealing identity of guest**~~
        ii. If Meal.anonymity = false: notifies guest user and host user of their identities and matched location and time
4. Transfer_Meal(host: User, guest: User)
    a. Req: None
    b. Effects:
        i. Call MIT Dining API to transfer meal from host to guest
5. **Database_Cleanup(string currentDate)**
    a. **Req: None**
    b. **Cleansup expired requests given currentDate**

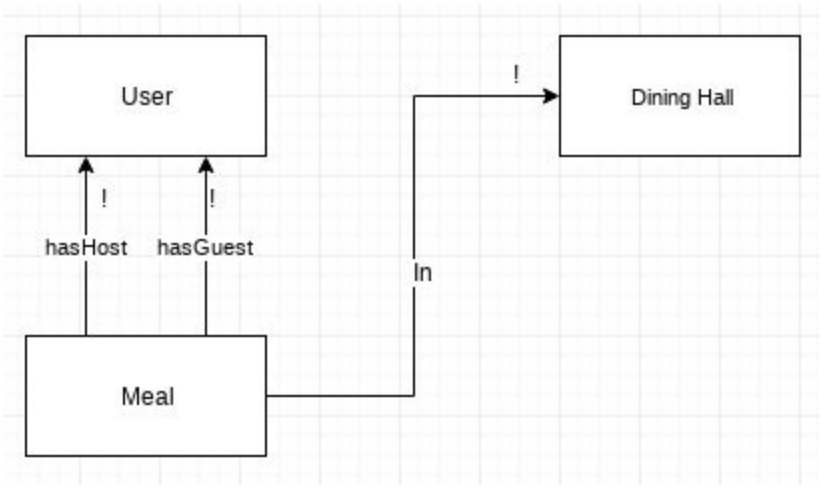
d. Operational Principle

User A who suffers from food insecurity will make a ~~**anonymous**~~ guest request on our application for a meal. User B is also food insecure and makes a guest request by setting time interval and dining hall location availabilities indicating willingness to meet-up. User C who has excess meal swipes donates two meals indicating willingness to meet-up by providing time interval and dining hall location availabilities. ~~**User A receives the meal and is able to use it to swipe at any time or dining hall due to their anonymity**~~. Users B and C receive a meet-up meal with a time and dining hall location, where they meet and have a meal together.

2. Meal Tracker
   a. Purpose

      To allow users to keep track of how many meals they have donated

   b. Data Model



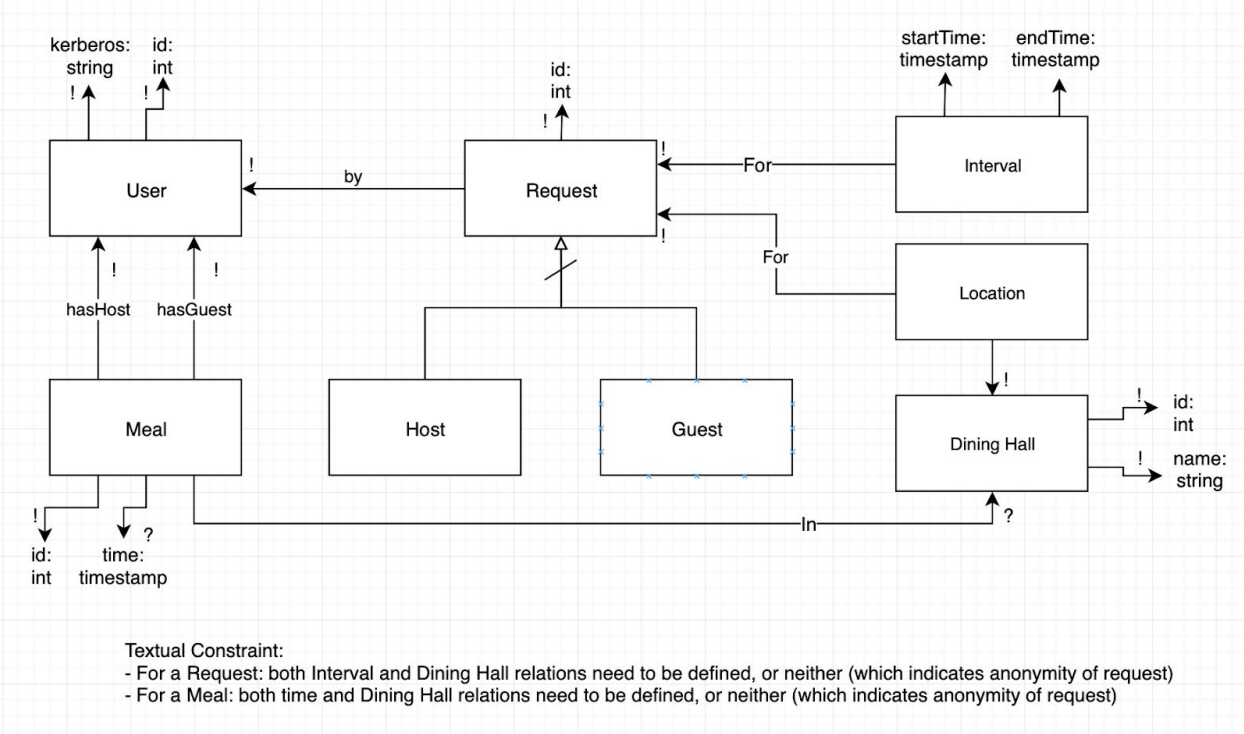
   c. Behavior
      i. User actions
         1. Donate_Meal(user: User)
            a. Req: None, user already confirmed student
            b. Effect: the user becomes a host for the transaction, and may not be anonymous
      ii. System Actions
         1. Add_Meal(guest: User, host: User)
            a. Req: None
            b. Effect: the tracker increments number of donated meals by one
   d. Operational Principle

      When a user donates a meal, and they successfully complete the transaction, the tracker will count that as another meal donated. This graphic is then displayed to the user so that they can keep track of how many meals they have donated, and how many they have left.

Note: We do not have a concept for Authentication since Kerberos API takes care of that (and hence did not list all the user actions such as create_user, delete_user, login, logout, etc) because they are not part of our purposes although they will be actions available to the user and follow a similar design as the ones implemented in previous assignments.

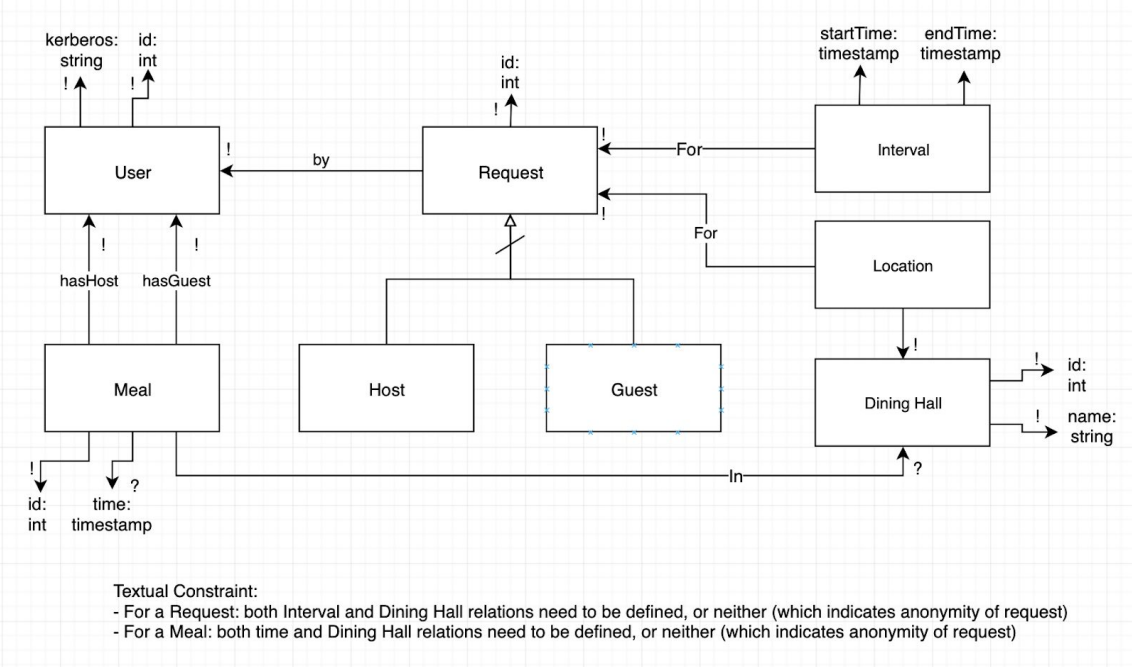
Data Model

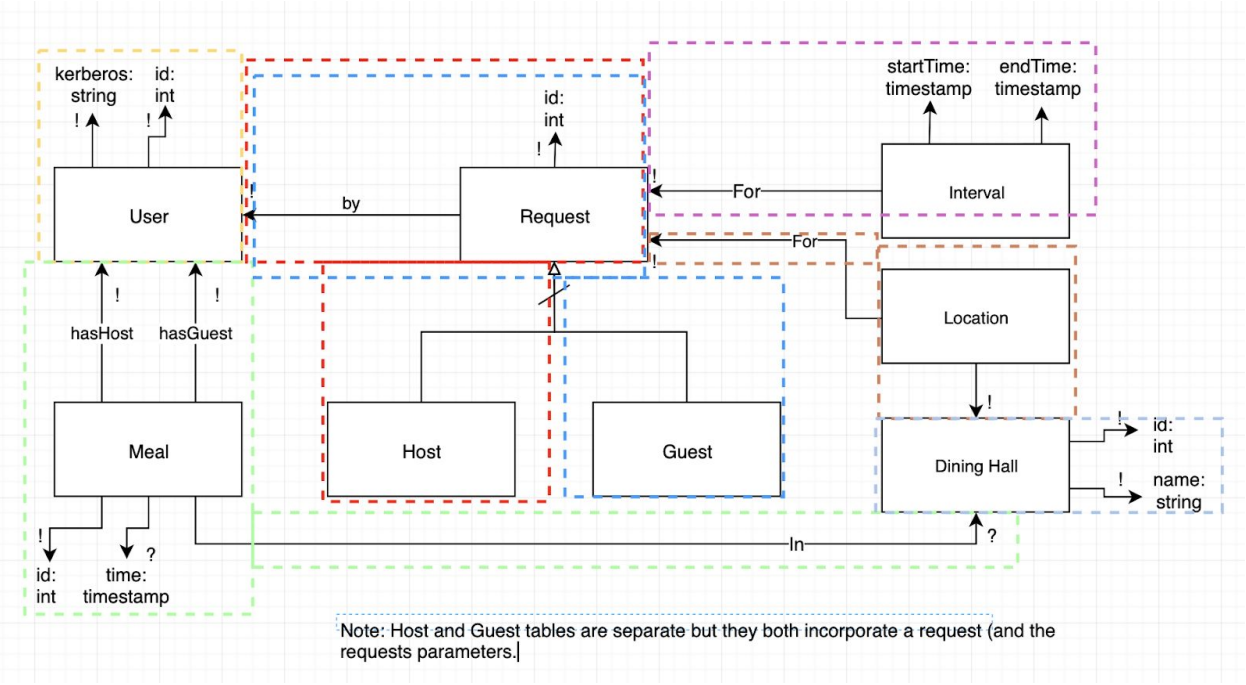
You should provide a data model that brings together into a single model all the models from the concepts, and any additional data that is needed (e.g. for simple CRUD purposes).



Schema Design

Provide a definition of the tables that will comprise your database. You can use SQL code to do this. You should include an explanation of how the schema was obtained from the data model, making intermediate steps clear.





Users:

[ID | Kerberos | Password]

```
CREATE TABLE IF NOT EXISTS `users` (
  `id` varchar(30) NOT NULL,
  `kerberos` varchar(30) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Meal

[ID | Guest | Host | Time | Location]

```
CREATE TABLE IF NOT EXISTS `Meal` (
  `id` int(255) NOT NULL,
  `guest_id` int(12) NOT NULL,
  `host_id` int(12) NOT NULL,
  `time` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `location_id` int(12) NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Interval

```
CREATE TABLE IF NOT EXISTS `Interval` (
  `req_id` int(255) NOT NULL,
  `start` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `end` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`req_id`,`start`,`end`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Host_Request

[ID | User | Time | Location]

```
CREATE TABLE IF NOT EXISTS `Host_Request` (
  `id` int(12) NOT NULL,
  `user` int(12) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Guest_Request

[ID | User | Time | Location]

```
CREATE TABLE IF NOT EXISTS `Guest_Request` (
  `id` int(12) NOT NULL,
  `user` int(12) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Location:

[ID | name]

```
CREATE TABLE IF NOT EXISTS `Locations` (
  `id` int(30) NOT NULL,
  `name` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Dining_Hall

```
CREATE TABLE IF NOT EXISTS `dining_hall` (
  `req_id` int(11) NOT NULL,
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  PRIMARY KEY (`req_id`,`id`,`name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Security Concerns

This section addresses the potential security risks of the app and how your design will mitigate them:

- A summary of key security requirements: what might go wrong and why it matters
- How the security risks will be mitigated, especially those involving standard web attacks (e.g. XSS, CSRF, etc).

Problems that could happen:

- Data integrity / Integrity Violations - we are holding people's personal information and we need to make sure that people are only allowed to sign up to this up if they are MIT students. This is inherently an important concern that we must mitigate.
- Protect against SQL injection - We will be interacting with SQL tables very widely in our application, therefore this will be a requirement.
- XSS - again, lots of SQL tables interaction and users often have chances to input strings, hence could inject code in their inputs
- CSRF - enough said

Protective strategies:

- Data Integrity / Integrity Violations: we will make sure that all authentication information is secure by using MIT's Certificate/Kerberos authentication (Duo authentication). This will also make sure that people who use the app are MIT students
- Prevent against SQL injection by using the database's escape capabilities wherever there is a query being made
- XSS can be easily prevented by using Vue's double brackets which prevents the injection of code into the web application. This will successfully sanitize all of the users' inputs which become part of the webpage for other users. We can also use the DOMPurify() library
- CSRF: CSURFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF it!!

Wireframes

https://app.moqups.com/fradavelletri1@gmail.com/Qu53ifa8at/view

Paths to Take

1) ~~**Donate an anonymous meal**~~
    a) ~~**Log In**~~
    b) ~~**Click Host a Meal**~~
    c) ~~**Click Donate button after having filled out the form**~~
2) Host a meet-up meal
    a) Log in
    b) Click Host a Meal
    c) Click Host button after having filled out the form
3) Request to join a meet-up meal
    a) Log In
    b) Click Join a Meal
    c) Click Join button after having filled out the form
4) ~~**Request an anonymous meal**~~
    a) ~~**Log In**~~
    b) ~~**Click Join a Meal**~~
    c) ~~**Click Request Anonymously button**~~

Note: After trying one of these paths please revert to the login page so as the state of the number of donated meals makes sense.

Note: We named the button Join A Meal instead of Request or Get A Meal because we wanted to avoid the potential stigma of asking for a meal. Although we did consider how this might not be the clearest language to use in terms of UI/UX heuristics. We are open to suggestions :)

Design Commentary

As you worked on your design, you will have considered many tricky questions, and will have made decisions between alternatives, sometimes making a tradeoff choosing one imperfect solution over another. This section of your design document brings together all the insights you have gleaned about your design -- why did you settle on the design that you did? It can be presented as a collection of footnotes on the rest of the design document, or as its own free standing text. But however it is expressed, it should discuss substantive and interesting questions, and will be a significant influence on the grade you receive for your design.

Authentication:

We discussed how we wanted to perform the creation of an account and signing in. This is a very important issue for 2 reasons; we must keep user's integrity to make sure that the user signed in is who they claim to be, and we must confirm that the account is controlled by a student. The second point was the most interesting part to discuss. We decided between doing email confirmation or using MIT certificates. Email confirmations would ensure that the user had access to an MIT email, however, that may not prevent alumni or people with access to an MIT mailing list from using our app. For that reason, we decided to use MIT certificates for authentication purpose.

~~How to deal with Anonymity:~~

~~This was a particularly difficult question that we struggled with. We wanted to give the guests the option to receive meals anonymously, however it was a trouble to incorporate that into our design. To attack this problem, we decided to allow guests to choose either anonymous or meet-up, and we allowed hosts to choose just anonymous, or either one. This way, we optimize the amount of matches to be made, while also allowing for anonymity. When the user chooses to be anonymous, a place and time is not required. Therefore, anonymity can be deduced by the presence of a time or place in the meal and request tables, and this is the reason that we did not include anonymity in the SQL tables.~~

Locations:

To keep with normal form for our tables, we decided to create a table for the dining halls that each request is for. Therefore, when a user selects dining halls that they are willing to go to, each selection and the request id becomes a new row in the dining halls table. In addition, we created another table that serves as a list of the dining halls so that once a location is finalized for the meal, it can point to this single location rather than just storing it as a string.