Revised Design

Motivation

Brief Description:
Eat, Meet, MIT is a web application designed to allow students on campus to meet other students through MIT's dining halls. Students fill out a request form indicating which dining halls they'd be willing to eat at, and what time. Our server then matches students together according to their request.

Key Purposes and Problems Solved:

- Students want to eat at new dining halls, but don't know anyone there. Our app addresses this by allowing students to list which dining halls they'd be willing to eat at.

- It can be hard to plan to eat with company. Our app addresses this by matching students with one another for a set time and place.

- Making new friends becomes harder as the years progress. Our app addresses this by including a Beaver Network that lets them maintain friendships with others they've met through our app.

Deficiencies of Existing Solutions:
Facebook is too broad of a platform, and it isn't designed to meet new friendships (but rather maintain old ones). Alternative is to just go to new dining hall and introduce yourself to someone new, but social norms inhibit this type of behavior (e.g., people tend to be shy).

Concepts

Request

- Purpose - Enables users to fill out dining preferences and availability for system to match.
- Operational Principle - If you want to get matched with someone for dinner at a dining hall, you fill out a request form with your preferences and availability.

Status

- Purpose - Allows user to manage their current request.
- Operational Principle - After you send a request, the status on your home page will update (originally set as “No Request Sent”). If you’ve gotten a match, it will show “Matched” and also show you the user you are matched with; from here, you can then choose to either cancel or confirm. Otherwise, it will say “Pending.”

Network

- Purpose - Allows users to keep track of other users they’ve connected with through our app.
- Operational Principle - After dinner you will be prompted if you want to add the other person to your Network (as will they). If both of you accept, you will be connected.

Conversation

- Purpose - To communicate with a person you’re connected to.
- Operational Principle - If you want to communicate with someone in your Network, you can send each other messages through your conversation.

Misfit(s)

- What if a user wants to change/cancel their request after they’ve filled out a form?
    - We allow users to cancel or change a request after submission and before a match.

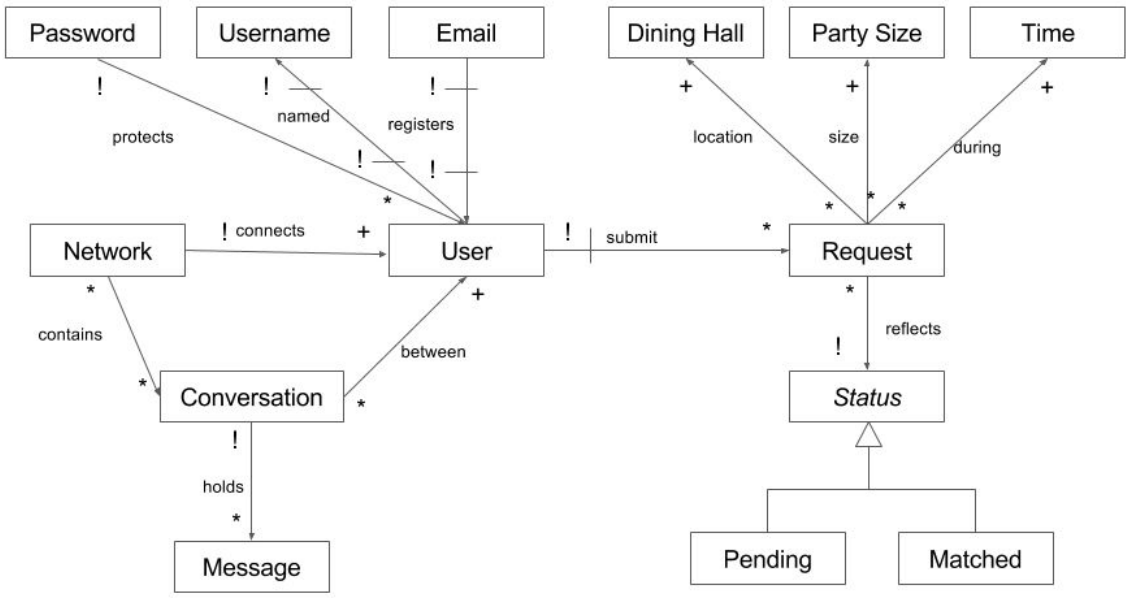
Data Model

Textual Constraints:

- A conversation has exactly two users.
- A user can only submit one request per day.

Design Insights:

- We had to think about the different statuses that a request can have, and whether they can change, and how often. Thinking about this brought certain risks to light, such as whether a user changes his mind after being matched with someone.
- We also had to think about the number of users a conversation can hold, and whether we want to limit that to binary relations or not.



Security Concerns

Security requirements:

Our application doesn't afford the user to store any personal information, but there are some things stored that we aim to secure: a user's password, the match-up between two users, a user's network and the conversations in their network. Specifically, our application will be implemented such that the aforementioned items are kept private to the user(s) involved.

Aside from this, since we allow users to input text, there is potential for a cross-site scripting attack, which will work towards mitigating. Also, as our App only allows current MIT students to use it (we verify @mit.edu with confirmation emails), we don't have to worry about non-student individuals signing up and posing any threats.

***Threat Model*:**

User without credentials can only access Registration/Login page. User with credentials can potentially construct requests inside a conversation. Application doesn't hold any sensitive information, so highly unlikely we will be targeted by a skilled adversary.

***Mitigation Techniques**:*

- Brute force guessing at a user's password
    - Limit the number of tries a user gets
- Rainbow table attack
    - Store salt along with hash of password in credential database
- Cross Site Script attack in conversation
    - Sanitize user input

Design Challenges

We listed a number of solutions for each challenge. The solution(s) we ultimately went with appear in bolded text and have a reasoning between the pros and cons.

- Sometimes dining halls aren't open (holidays, summer, etc.). We don't want people to sign up for a time when a dining hall is closed.
    - ***We will try to look at the schedules and anticipate the major times when this happens (i.e. summer and IAP), and we'll manually disable our service during these times.***
        - ***Pro: Solves the issue in a relatively low intensive manner.***
        - ***Con: Could forget to disable said days.***
        - ***We picked this option because there are only five dining halls, and disabling the service is trivial***
    - Sync our app with MIT dining's calendar so that the app will automatically disable those dining halls on those dates. Contact the dining peoples for said calendar.
        - Pro: Automated and sleek.
        - Con: Could be difficult to pull together.
    - Have a report button so users can report when the dining hall is closed.
        - Pro: Self-regulating (by the users), especially the users who lives in the same building of that dining hall
        - Con: Extra implementation time and extra time to validate users' inputs
- What if a user decides they want to use our app in the morning, sends a request, but then forgets about it?
    - ***We'll have it so that users can rate other users, so users get penalized for forgetting***
        - ***Pro: Puts the work on the users, not administration***
        - ***Con: Relies on other users to rate one another***
        - ***Picked this option because it is a nice social feature***
    - ***We'll send then a reminder an hour before their scheduled meeting time.***
        - ***Pro: Reduce the chance of a user forgetting***
        - ***Con: Implementation might not be trivial; sending a reminder doesn't guarantee the user will see it***
        - ***Also picked this option because it makes our system user-friendly***

- Some dining halls close and open earlier than others.
    - Restrict the available times users can pick to the times when all dining halls are open
        - Pro: Solves the issue in a simple fashion
        - Con: Limits the use of our app
    - ***We'll use listeners and see which dining hall options the user has chosen. Depending on this we'll disable certain dinner times.***
        - ***Pro: Ensure user will not select a time that is not available***
        - ***Con: Need to implement additional features on each request form, and the logic and implementation might not be trivial***
        - ***We picked this option because it gives the user more functionality from our app***
- Users sends a request but doesn't get matched with anyone.
    - **If one hour before the chosen date the user is not matched with anyone, we'll send him suggestions of other times/dining halls that he could edit his request to that would match him with someone.**
        - ***Pro: This feature introduces flexibility to the users and increase their chances of being matched with others if users are willing to change their dinner time or dining halls.***
        - ***Con: We need to wait for the response from users and that may delay the matching and potentially block the progress.***
        - ***We picked this option because it increases the chances for a user to get utility of our app***
    - Our algorithm could try a "best fit" approach and potentially match a user at a time/location that they didn't originally choose, and ask for their confirmation
        - Pro: Feature would provide a lot more matches in the first round
        - Con: Difficult to implement, and also defeats the purpose of matching according to availability/preference

Design Choices:

- Originally we had both a page listing a user's "Network" as well as an "Inbox" with the user's conversations. After some thought we decided to get rid of the "Inbox" feature since it would be too similar to the user's "Network" and just moved conversations to the user's "Network" instead.
- Originally we gave the users the option to look at requests that have already been chosen, and then see if they would like to pair up with any of them. Also our original implementation included profiles. We decided to move away from this and instead handle all the matchings on our end instead. We did this because we

thought the other choice went against our original idea. We wanted users to be able to meet new people through our app, and letting users make choices might lead to them only meeting a certain kind of people, and not actually expanding their pool of friends.

- Another thing we had to think about was whether we wanted people to be able to keep track of who they connected with through our app. We decided to do this for a variety of reasons. First of all, this would allow users to enhance the connections they've made through our app, and actually develop friendships with these people. We also decided it would be better if we didn't match two people who already have been matched together before, so our implementation doesn't allow matches between people who have each other added in their "Network".

User Interface