Updated Full Design

Overview

SwipeBuddies is a service unique to MIT that allows students to donate extra meal swipes. Since the app does not have access to student's meal plan accounts, swipe givers & receivers need a specific time/date and dining hall where the giver physically swipes in the receiver. With this strategy, SwipeBuddies aims to promote food accessibility and reduce food waste, as well as to encourage students to socialize and meet new people.

Currently, MIT has a swipe-sharing service called SwipeShare, but there are limitations that prevent it from being applicable to the entire student body:

1. a student can donate at most six swipes
2. a student has to apply & qualify for the program to receive swipes.

In contrast, SwipeBuddies has no donating limit, doesn't require any application, and donating/receiving a meal is as simple as filling out a three question form. Another current alternative is guest passes, which allows student to feed another student free of charge.

SwipeBuddies will feature:

- A scheduling system to track user's time/location preferences and match history
- A matching algorithm to maximize the number of swipes shared and optimize the distribution of meals given the list of swipe donors, availability times, locations, and user reputation

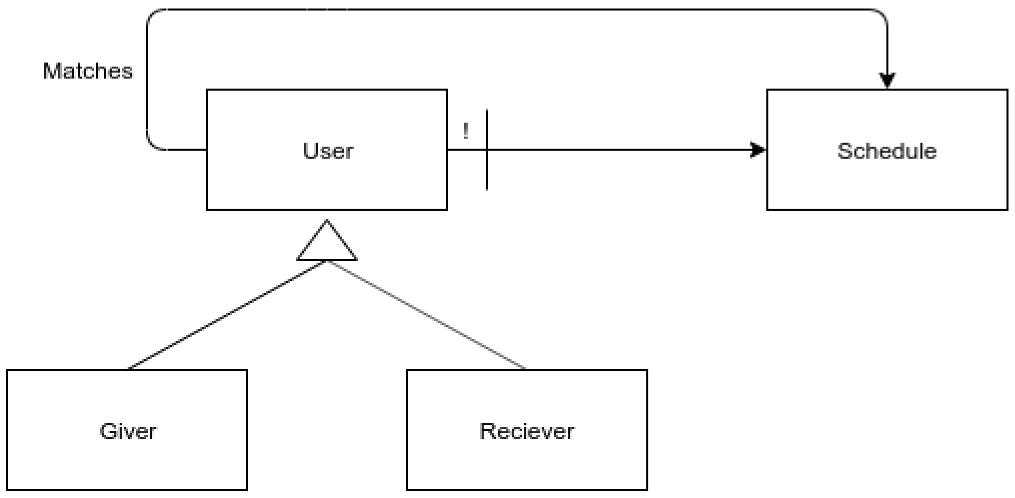
Since our matching algorithm prioritizes creating the maximum number of matches based on user's schedule (via a weighted bipartite matching algorithm), it fulfills our mission of maximizing the number of swipes used.

Conceptual Design

Concept: Matching

Purpose: to maximize the number of matched swipes while maximizing the number of receivers

Structure:



Behavior:

```
createMatches (s: Schedule):
    u->matches = MatchResults;
```

Tactics:

```
createMatches(s:Schedule);
```

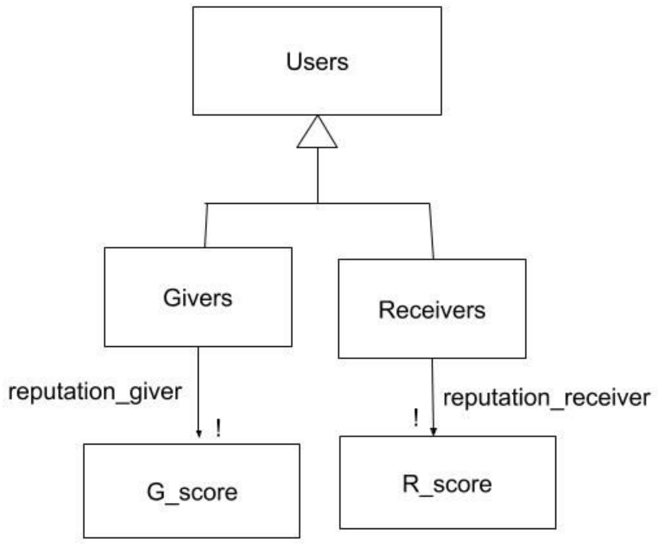
Comments:

MatchResults comes from our matching algorithm that runs everyday. It matches givers & receivers based on their schedules.

Concept: Reputation

Purpose: to discourage users from flaking. A receiver’s reputation score is used in the matching algorithm. Low score means low chance of getting a match. A giver’s reputation score will be shown to all users.

Structure:



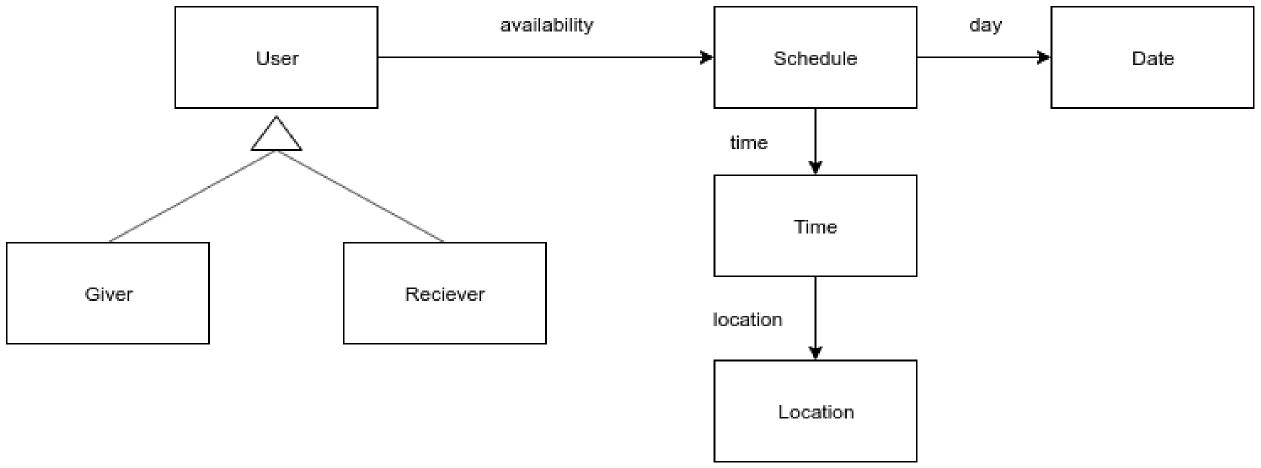
Behavior:

```
increaseReceiverScore(Receiver: r, Int: i):
    r.reputation_receiver = min(r.reputation_receiver + i, 100)
decreaseReceiverScore(Receiver: r, Int: i):
    r.reputation_receiver = max(r.reputation_receiver - i, 0)
increaseGiverScore(Giver: g, Int: i):
    r.reputation_giver = min(r.reputation_giver + i, 100)
decreaseGiverScore(Giver: g, Int: i):
    r.reputation_giver = max(r.reputation_giver - i, 0)
```

Tactic:

If a receiver did not show up, decreaseReceiverScore(r, 10).
If a receiver showed up, increaseReceiverScore(r, 3).
If a giver did not show up, decreaseGiverScore(g, 25).
If a giver showed up, increaseGiverScore(g, 5).

Concept: Schedule
Purpose: To allow users to request/give at a specified time
Structure:



Behaviors

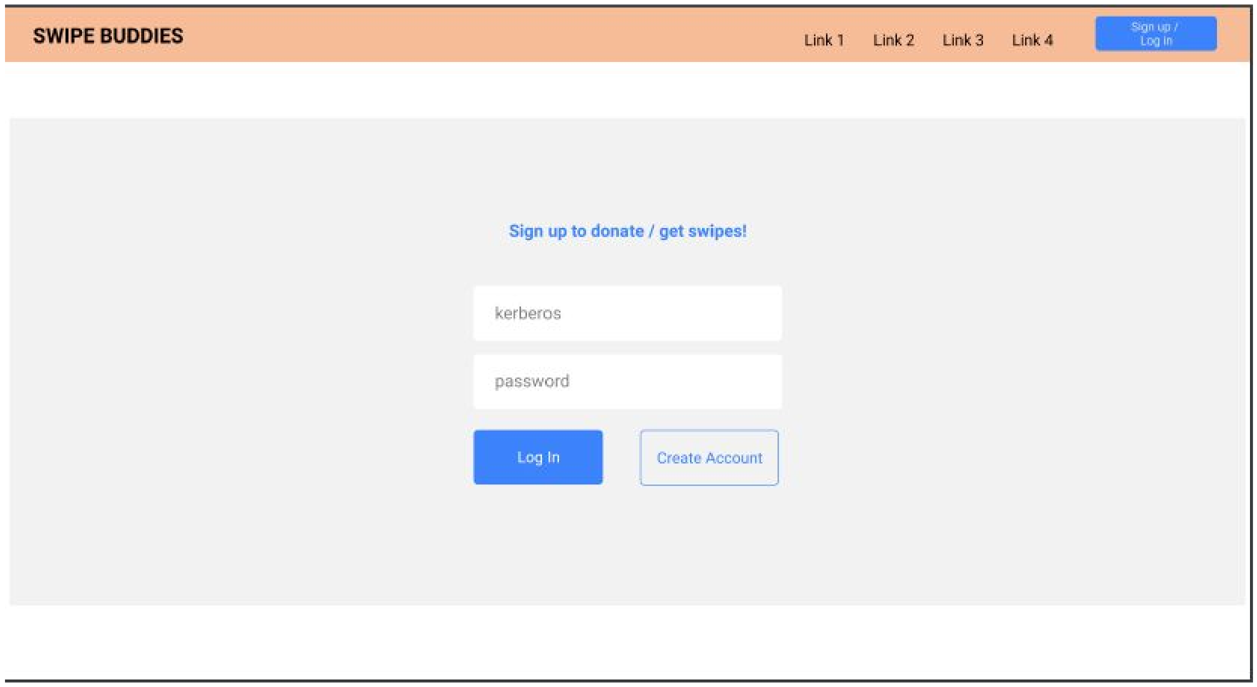
```
MakeRequest(u: User, d: Date, t:Time, l:location):
    s.day = d; s.t = time, s.t.l = location;
    U.availability += s;

EditRequest(u: User, d: Date, t:Time, l:location, d':Date', t':Time', l':Location'):
    DeleteRequest(u: User, d:Date, t:Time, l:location);
    MakeRequest(d':Date', t':Time', l':Location');

DeleteRequest(u: User, d:Date, t:Time, l:location);
    U->availability -> days -= d;
    u->availability -> time -= t;
```

Tactics:

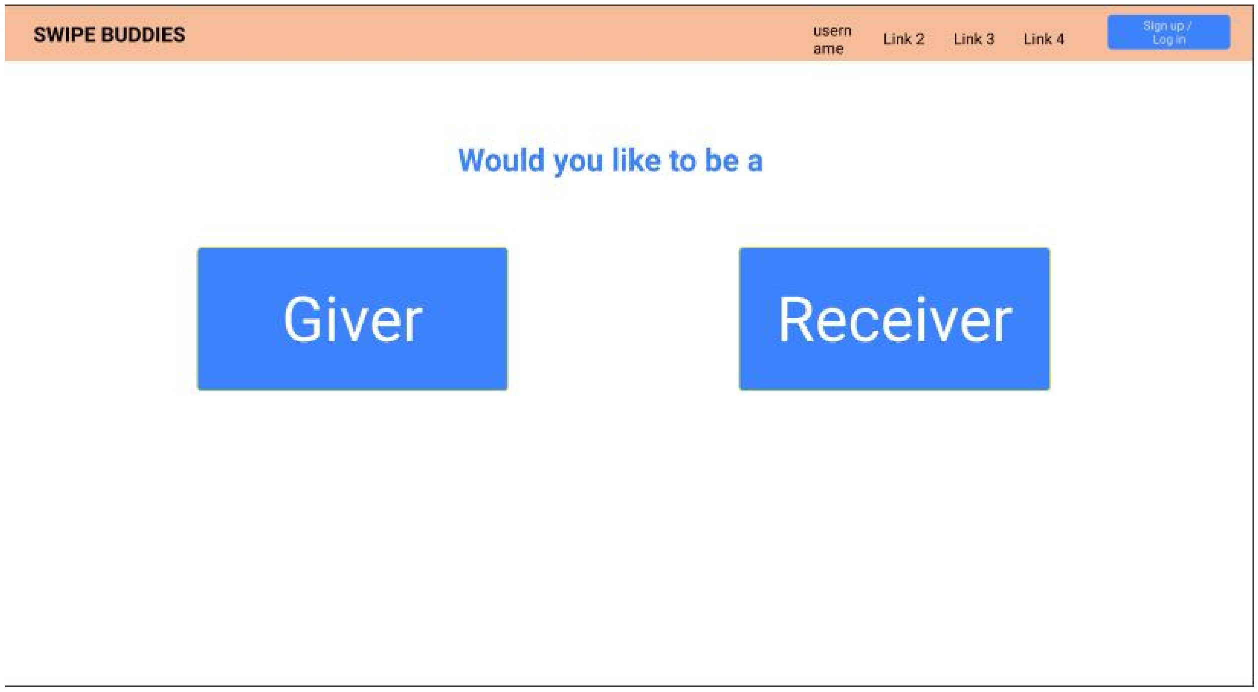
MakeRequest() to give or request a swipe.
EditRequest() to modify a schedule.
DeleteRequest() to delete a schedule.

Initial Wireframe

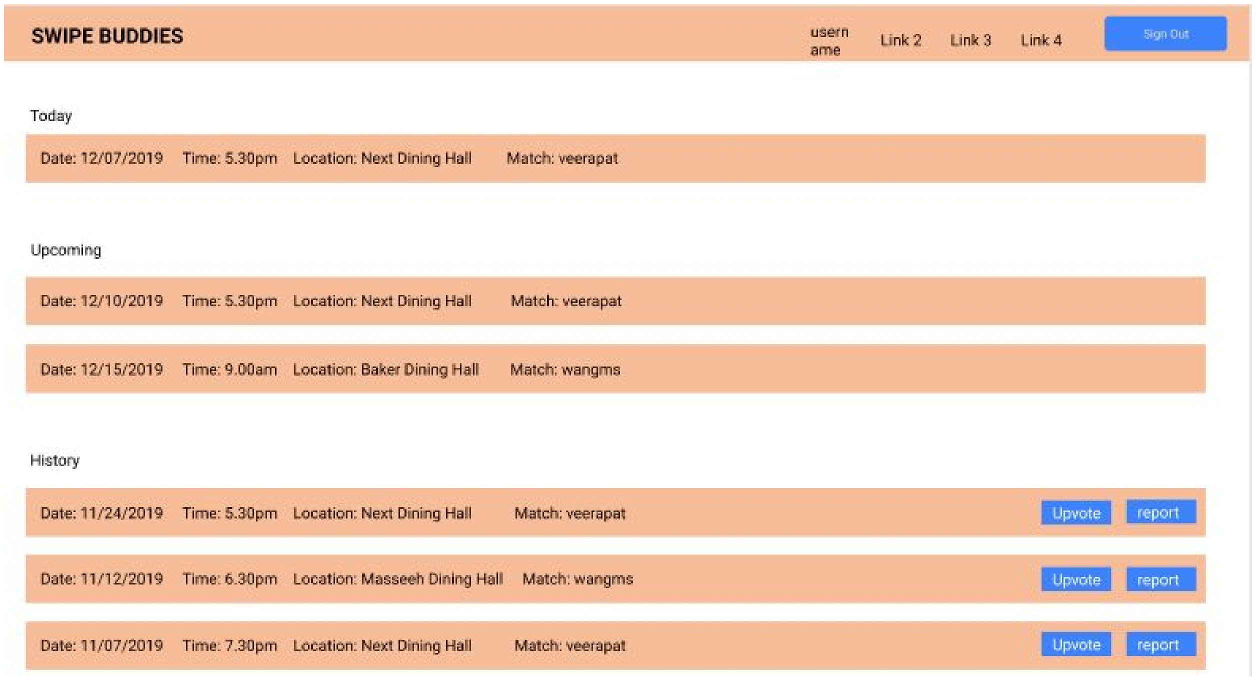
Welcome Page



Create Account Page (before continuing to kerberos)



User Profile Page (NavBar -> username -> links to Profile page)



Today

Date: 12/07/2019 Time: 5.30pm Location: Next Dining Hall Match: veerapat

Upcoming

Date: 12/10/2019 Time: 5.30pm Location: Next Dining Hall Match: veerapat

Date: 12/15/2019 Time: 9.00am Location: Baker Dining Hall Match: wangms

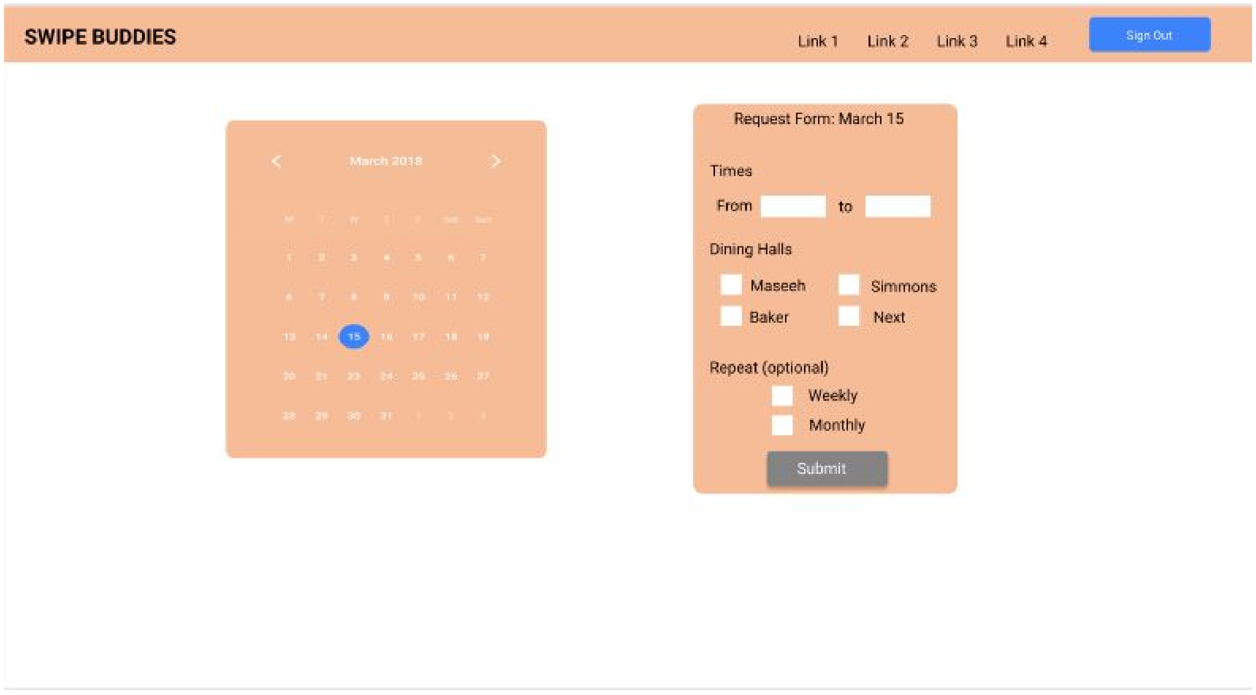
History

Date: 11/24/2019 Time: 5.30pm Location: Next Dining Hall Match: veerapat Upvote report

Date: 11/12/2019 Time: 6.30pm Location: Masseeh Dining Hall Match: wangms Upvote report

Date: 11/07/2019 Time: 7.30pm Location: Next Dining Hall Match: veerapat Upvote report

Calendar Page (Logo -> links to calendar page)



Request Form: March 15

Times

From [ ] to [ ]

Dining Halls

[ ] Maseeh [ ] Simmons

[ ] Baker [ ] Next

Repeat (optional)

[ ] Weekly

[ ] Monthly

Submit

Design Commentary

System Matching

Alternative: First come, first served

At first, we planned to build a first come first served system where givers' schedules are shown to receivers, and receivers can claim a schedule to make a match. However, in a case that there are many receivers, then some receivers might keep missing a match. Considering the goal that we want to distribute meals to as many receivers as possible, we abandoned the idea.

In the end, we let the system do the matching. Givers and receivers specify the date, time, and location in a form, and submit it to the system. The system collects a pool of givers and receivers, then does the matching, and returns results to givers and receivers on the day before the meal. In a case that there are more receivers, then the matching algorithm will consider a receiver's reputation score, the last day they received a meal, and the number of meals they have received so far over a semester. On the other hand, if there are more givers, then givers will be sorted by their reputation score, and those in the top will be matched first.

Reputation / Feedback

We decided to implement a rating system because we think it will help maximize the number of donated meals. Consider a situation where a receiver had failed to show up during a meeting. Failing to meet up means that another person who could have received the meal, but did not. We prefer to give the meal to the student that will show up, thus increasing the average number of people who get a free swipe.

Additionally, we decided to create two separate scores: a giver score and a receiver score. We do this because the different natures of giver vs receiver. (e.g. a receiver failing to show up may not hurt the giver as much since the giver can still go into the dining hall, but if a giver fails to show up then the receiver starves. See section below: Rating users differently for more on this.)

One concern is that the reputation system may deter givers from donating swipes since they may not want to be rated. So we decided that the rating system will only include a reputation system like StackOverflow's (gold, silver, bronze) and not a rating system like Uber’s (where user is rated with a number). This way, givers are not visibly penalized. Ultimately, if the reputation system did deter people, it ideally deters those who are uncommitted (those who stand up their matches repeatedly).

In all, we believe a reputation system will bring more benefit than harm. Consider a case where no such rating system was implemented. This means uncommitted receivers or givers are matched with more committed users, which reduces the number of swipes given because a matched swipe does not go through if either side of the match fails to show up.

Rating users differently

We decided to weigh reputation differently based on user type (giver vs receiver). When a giver shows up, they are awarded a higher score compared to a receiver because of the beneficial nature of the giver. When a giver no shows, they are also penalized more than the receiver because it has a more severe consequence (the receiver who get stood up goes hungry).

When the givers outnumbers the receivers in the matching system, we will use the giver’s ratings to prioritize matching givers with higher scores.

Likewise, when receivers outnumber givers, we use receivers’ scores to give preference to better reputed receivers.

The alternative to this design would be to weigh givers and receivers scores equally, which would incur deadweight loss since donor and receivers contribute differently. Although it seems more fair to treat users equally, we choose to rate them differently to maximize utility to the MIT community.

Scheduling/Requesting/Giving

We decided to design a feature to keep track of user's time/location preferences and schedule/match history, basically anything that lets us store user form input data.

The Schedule set is designed to store all donations/requests throughout the app's lifetime, and to pose as a unified lookup system for finding these current/past request events. For example, to find the current time/location submissions for matching the very next day, simply filter the Schedule set for entries that match the current day.

In the user profile, each user has a list of previous people matched with and upcoming matches. We decided for users to access current matches to easily reminding them of events via email. Meanwhile, a user's match history is stored in the profile tab so that the user has a chance to rate/report past matches.

One alternative is to create a separate History set, where past matches and time/location preferences are stored separate from the Schedule set. There's a tradeoff between simplicity and (barely significant) performance, and we decided to go for simplicity.

Design Commentary (Updated)

Used Firebase as our Database

Firebase is a fast external service Database that supports JSON. Unlike SQL, which is typically vulnerable to SQL attacks/injections due to poorly written code, Firebase is more resistant to those types of attacks. Firebase additionally offers data validation.

Used OpenID

We originally chose to use MIT openID to make sure students who sign up are actually MIT students, and each user can only have one account (unless they make multiple kerberos - but that takes effort). We managed to connect to MIT openID; However, we are finding that the MIT openID api is sometimes slow (takes 10 minutes) and inconsistent (returns 500). Additionally we are experiencing technical difficulties in redirecting the session information from authentication to the user session. We may end up reverting to normal log-in if these issues persist.

Resigned Calendar UI

During implementing the calendar, we decided a Drag and Select system of time selection made more sense than our previous solutions. Its intuitive for MIT users most have been exposed to a When-2-Meet interface. We decided to color coat times based on dinner type as well, making it easier for users to understand what times they are selecting.

Additional Specificity in Selecting Locations

Calendar also includes selecting locations for individual time slots. These are done by selecting dorms a user prefers, then allow them to remove them from individual time slots for additional specificity. This is significantly better than in our old design that constrained the user to specify dorms for the entire meal (breakfast, lunch, etc).

Color Schemes

An important part of GUI design is to make the interface presentable. We learned in lecture that having color harmonies can help. We chose a color harmony of three colors for our design. This included blue, purple, and green. We added grey and white as other accompanying colors. The result made our design look significantly better than the last iteration.

User Confirmations

Initially we did not include user confirmation for certain actions. However, we changed this as we realized canceling a match is important. If a user accidentally canceled, they miss out on a meal. Our design can have real life consequences. Thus, many actions include a user confirmation box for actions. These include, canceling matches, switching roles, and reporting users.

Ethical/Social Reflection

Once in use, SwipeBuddies will affect direct and indirect stakeholders. Direct stakeholders include the users, who are the givers and receivers of meal swipes. Indirect stakeholders may include MIT Dining and SwipeShare. For MIT Dining, stats may be affected if enough people use SwipeBuddies to change the behavior pattern at dining (e.g. significantly higher number of guests, fewer unused swipes, etc). Likewise, if enough people use SwipeBuddies, it may cause fewer givers to donate to

SwipeShare, which may be undesirable as SwipeShare specifically assists a group of qualifying students.

A social issue that may arise with the use of SwipeBuddies is the power dynamic between the givers vs receivers. By nature of the receivers being obliged to the givers, there is a danger of the receivers being secondary class citizens. Especially since SwipeBuddies promotes social interaction between the giver and receiving by requiring a meetup, this discrepancy in social status between strangers could cause awkward interactions.

One possible solution to resolve this issue is to allow receivers to buy the swipes at a low price (e.g. $2/swipe). This 1) is still a cheap price for a meal to the receiver and 2) offers an incentive to the giver to give their swipes on the platform, where they can get some money back for swipes that they otherwise would waste.

Another app design alternative to alleviate the issue is to when the user fills out their request/give form, to ask them whether they want to socialize. The app then can match people who want to socialize with each other, and people who do not want to socialize together. This way people do not feel like they HAVE to socialize to give / get their swipe.

We chose our current design (without payment & without option where user chooses to socialize or not) because

1) it is an MVP that is simplest to implement and

2) the app becomes more complicated if we choose to involve money transactions

3) to adhere to the initial mission of providing free meals (and reducing food waste) while encouraging students to meet new people.

The alternatives listed above are certainly potential updates if SwipeBuddies goes into use and finds itself needing these changes.

After adding the reputation concept to our design, another social issue may arise is whether the system will make it harder for new users to join the community. This may depend on how we implement the reputation system (e.g. if we can distinguish between new users vs. users who just have average reputations).

If we assign new users with an average or low reputation (e.g. bronze or silver star), it may lower their chances of getting matched, since the matching system prioritizes users with higher reputations. And until new users are matched, they cannot demonstrate that they are committed users and raise their reputation. This becomes a paradoxical situation for the new user. Another site which demonstrates this situation is StackOverflow, where user actions(posting, commenting, upvoting) are limited until the user has enough reputation, but the user cannot gain reputation without performing actions. To avoid this situation, we may use user history as reference in addition to or as part of reputation.

Ethical/Social Reflection (Updated)

In addition to ethical reflection discussed in full design, some additional considerations are:

- The app lacks incentive for givers (other than, perhaps, friendship) - what if there are many receivers signed up but few givers?
- Any significant bugs in the system can potentially make a user miss out on a meal. Our app can prevent, or give people food simply by mistake.
- Our Rating System may be abused by others and potentially can be used as a way of increasing ones chances of getting a free meal. To defend against this, we need moderators to vet the reports.
- Administrators can abuse the system (having admin permissions) and take all the matched for themselves. We might have to add cryptography, or new permissions to defend against this.
- We are competing against SwipeShare. That MIT program gives swipes to people who need it (after 6 swipes), our program does none of that. This means that we might actually take swipes from people who may need it.
- The admins have access to the database (unencrypted) which means we have the entire list of students who are givers and receivers. If enough students register, this could potentially give us data that can be used to observe things such as relationship between student demographic & giving/receiving patterns, which may be an unintended side-effect of the app.
    - If students register with their kerberos & password, this also means we as admins have access to other student's MIT log-in information
    - One way to fix this would be to encrypt all data in the db. Then the only thing the admins know is analytics-related information (number of users, match frequency, etc.)

Nielsens’ Usability Heuristics (Updated)

1. Visibility of System Status
    a. Our application fulfills this heuristic. All submission buttons lead to some kind of information given to user. For example, if the user submits a schedule with nothing filled out, we inform the user that they must fill out said form.
2. Match Between System and the Real World
    a. The calendar itself is a real world object people are familiar to use, making it obvious to click a date. We use an arrow to indicate to moving through the calendar, a convention used everywhere.
    b. Phrases with the Login/SignUp pages are clearly understandable and visible to users, no jargon here.
    c. Submission form also uses clear language (ie Choose Dining Halls)
3. User control and Freedom
    a. For the submission form, if users make a mistake and submit their form, they can simply edit the form to fix their mistake. Given that the form asks for confirmation, we felt this was sufficient.
4. Consistency and Standard
    a. Our GUI conforms to most standards. The Login Page is similar to other sites, like Google and Amazon. Furthermore, our calendar follows standard convention, with arrows to switch through months. Scheduling information through a When-2-Meet like interface, which most MIT students have been exposed to.
    b. In Addition, language within our application is always consistent. We always refer to meals as meals, and times as times.
5. Error Prevention
    a. We use confirmation boxes to ensure that the user is doing the correct action. This is especially true if the user cancels their match, or submits a form.
6. Recognition rather than Recall
    a. Our calendar is similar to most calendars, clicking on a date reveals a submission form with a clearly stated date and title. The timeslots suggests to the user to click on the timeslots, like when2meet. We argue its easily recognizable interface.
7. Flexibility and Efficiency of Use

    a. For experience users we have an accelerator to help fill schedules faster. To fill out multiple consecutive time slots, the user can drag select multiple slots.

8. Aesthetic and Efficiency of Use
    a. Calendar is simplistic in design, no irrelevant information. The size of the calendar is also large, to help focus on what's important.
    b. Similarly, the History Page displays the minimal amount of information for a user to meetup with others. That is only the date, person, and time are shown. It is easily readable and understandable by a first time user.
9. Help Users Recognize Diagnose and Recover from Errors
    a. All error codes within our application are in plain English. These error codes can be seen when signing in with incorrect login information, missing sign up information, submitting empty forms. Any error codes that are not in plain English are unintentional.
10. Help and Documentation
    a. Currently we have no documentation within our application. However, we may benefit from it. Though its use would be for explaining what swipeshare is, and how it works.