6.170 RooMIT Design

Rujia Zha, Peinan Chen, Olga Shestopalova, Alexander Heifetz

Motivation

Most MIT students live in dorms (70%) and most of those students have roommates. MIT housing provides a way to lottery into each dorm, but no way to pair with a compatible roommate. RooMIT provides a solution to this problem by proving roommate suggestions to students according to their preferences- things that students usually find out about each other when they're well into their stay together.

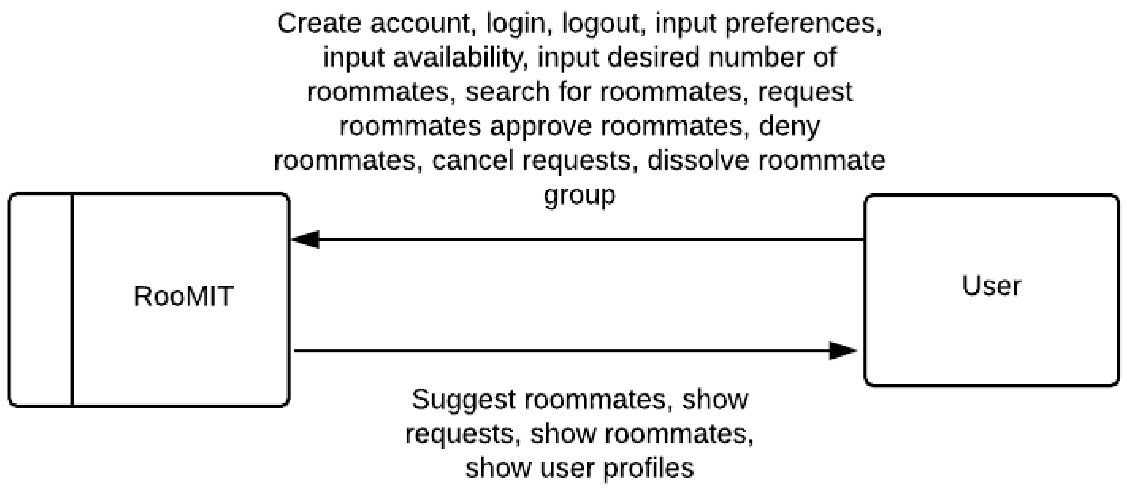
Students will submit their responses to an assortment of preference questions, and then get suggested users that have similar preferences and from there can request users to be roommates, and can get accepted or rejected. We support multiple roommate agreements (eg. triples, quads, etc). When users decide they are done with their group of roommates, they can declare themselves unavailable and remove themselves from the search system. Currently the app is limited to MIT students. We aim to eventually be able to communicate with other students via our app instead of email. RooMIT is not associated with MIT Housing.

Purposes

- **Help ensure roommate compatibility**. Currently the housing system does not have a formalized method to help students find suitable people to live with, so this application will help address this issue.

- **Help students get a comprehensive overview of their roommate options.** The idealized method for students to find the most suitable roommates would be to talk to everyone who will be living in their preferred residence(s). This app will alleviate the amount of effort/time it takes to do this by giving students access to everyone with similar preferences and desire for roommateship.

Context Diagram


Create account, login, logout, input preferences, input availability, input desired number of roommates, search for roommates, request roommates approve roommates, deny roommates, cancel requests, dissolve roommate group
RooMIT
User
Suggest roommates, show requests, show roommates, show user profiles

We are not using any external APIs, and have our own authentication system.

Concepts

- Preference
  - A preference is a statement that the user responds to. Each preference will have a description (the statement), and a response (‘Yes’, ‘No’, or ‘Don’t Care’). The user can only respond from these three choices; they will be presented as multiple choice inputs.
  - Each user will enter their responses to a set list of preferences, created by the owners of the system. All users have the same set of preferences and cannot modify, delete, or add preferences.
  - There are two special types of preference - a Dorm Preference and a Roommate Number Preference. In our suggestion algorithm, users must have preferences of these types that are not mutually exclusive to be suggested to each other.
  - The default response to each preference is ‘Don’t Care’. Therefore, when a user creates their account, their preferences will all be filled with ‘Don’t Care’ responses, so the suggestion algorithm will always have responses to work with.
  - The preferences of each user are visible to any user that views their profile so they can see clearly where they match and where they do not.
- Suggestion

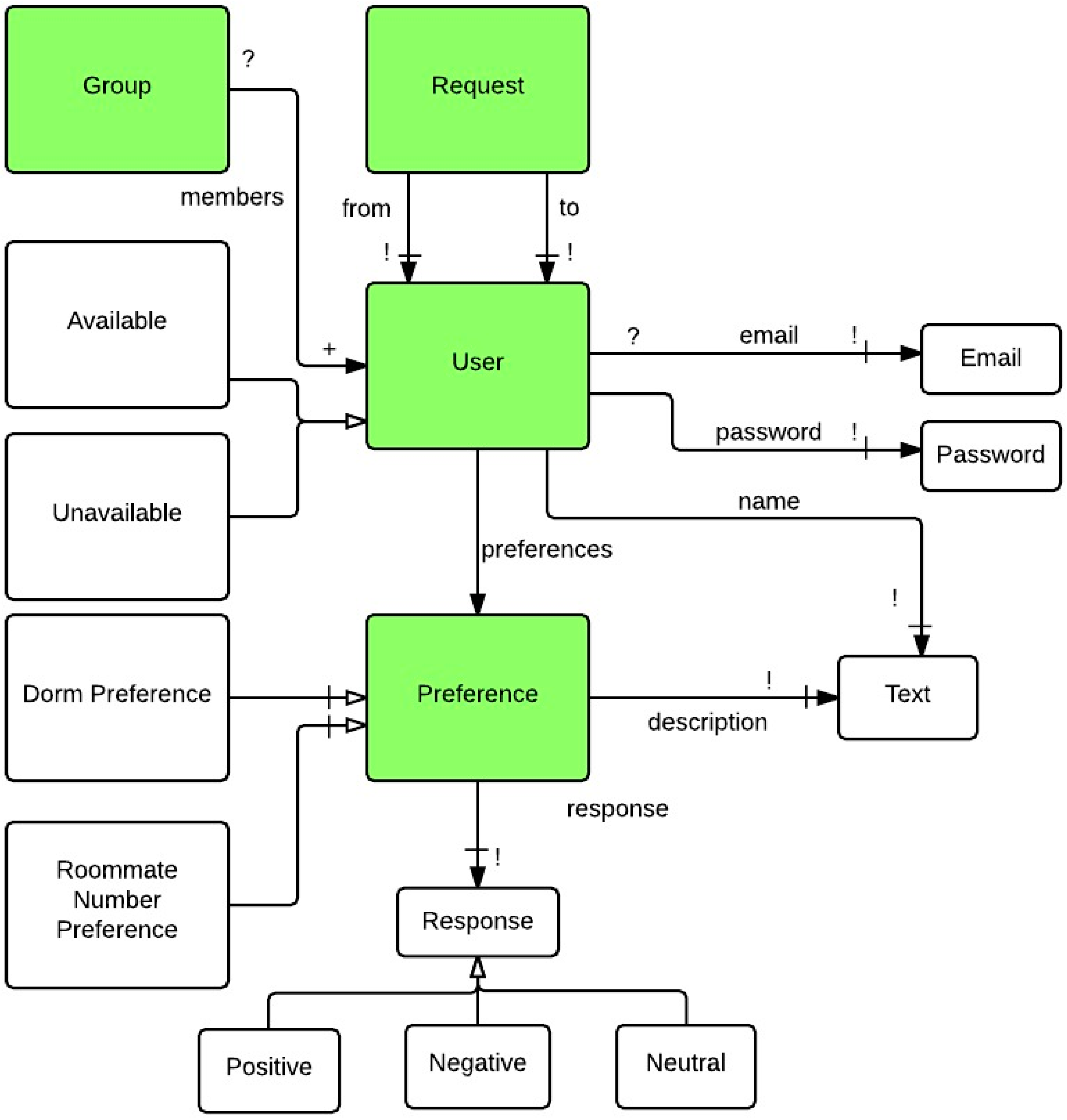
    - RooMIT will provide roommate suggestions based on how well the users match in terms of their responses to preferences. Compatibility is represented by a percentage.
    - If a user is unavailable, they will not see any suggestions because they marked that they are not looking for any roommates. The user will also never be suggested unavailable users. (See full suggestion algorithm below)
    - Compatibility is only checked between pairs of students. This means that if a user is looking to join, for example, an existing group of 2 users (and they are looking to be joined), the user will see their compatibility between each of the users, not an aggregate score.
- Request
    - A request is one user alerting another user that they wish to room with them. The requested user will be alerted and can either accept or deny the request. Upon accepting, the two become roommates. The user doing the requesting can cancel their request at any time before the requested user responds to the request (with an accept or deny). Once the request is accepted, any user may leave a roommate or group at any time.
    - When a request is sent to a user who already has roommates, the request is sent to the user and all of their roommates, and everyone must approve the request before the user requesting can join the group. Requests can be sent to any user with any number of roommates unless they are marked as unavailable. This means that if, for example, user A sends a request to user B who has 2 roommates: user C and user D, user A is proposing a quad (i.e. a group of four people) consisting of user A, user B, user C, and user D. Users B, C, D have to all confirm user A, and then the quad will be formed.
- Group
    - A group is a set of users that are roommates with each other. Each group will have a list of users that are in the group.
    - When a user accepts a request from another user, they are put in a group.
    - Every user in a group shares the same availability. If a user in a group is unavailable, everyone else in the group is also unavailable.
    - Users can choose to leave a group, which causes them to be removed from the list of users. When a group has 1 or less users, they will be removed and the group destroyed.
    - A user may only be part of one group.
- ***Changes:*** Group is now a concept. Dorm Preference and Roommate Number Preference are now subtypes of Preference.

Suggestion Algorithm

1. Initialize a list of suggestions that consists of all *available* users except for the current user.

2. Remove all users whose set of Dorm Preferences are mutually exclusive with the current user (i.e. there is no dorm that neither user has said “no” to). For example, if user A has only said “yes” or “don’t care” to Baker and Maseeh and “no” to everything else, and user B has said “no” to Baker and Maseeh, the users will not appear in each other’s suggestions.
3. Assume that the current user is User A. For each user B that hasn’t been removed by step 2, compare the Preferences of user A and user B. Pair up all Preferences of A and B by description. Assign each preference-pair the following score:
    a. If both users answer yes or both answer no, give +2 (reasoning: agreement is good)
    b. If both users answer “don’t care”, give +1 (reasoning: two users not caring about something is a good sign but not as good a sign as both of them agreeing one way or the other)
    c. If one user says yes or no and the other user says don’t care, give 0 (reasoning: one user strongly believing something and the other user not caring about it has no bearing on their compatibility as it won’t affect their relationship as roommates)
    d. If one user answers yes and the other user answers no, give -2 (reasoning: two users disagreeing about something is bad and is a signal that they are less likely to be compatible).
4. Normalize the score to a 0-100 scale (where all -2s are 0% compatible and all +2s are 100%) and assign each user B their corresponding score. Sort the Bs by score and return the list of user-score pairs to A as a list of their suggested matches sorted by compatibility.

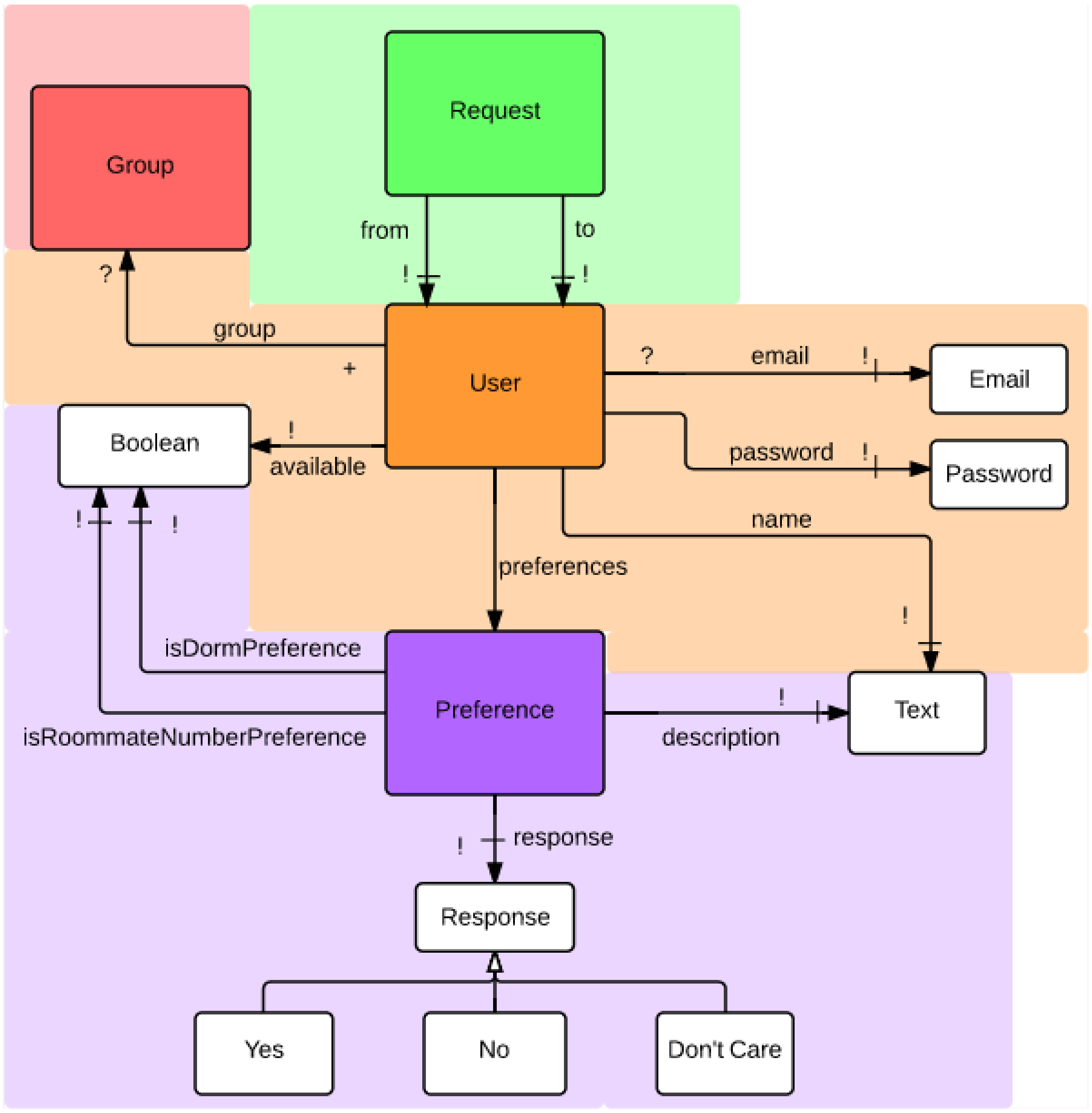
Data Model



- **User** is a student. A User can be available (looking for roommates) or unavailable (no longer looking, taken out of the search system). Users are created upon registration, and never deleted.
    - email: unique MIT email
    - password: password
    - preferences: list of Preferences of the user
- **Preference** is a User's opinion on a living condition. Users are suggested to each other based on how similar their Preferences are. There exists a predefined set of all Preference objects (each is a unique combination of description and response). When a User sets or changes a Preference, they change their reference to an existing

Preference object. A User cannot add or delete Preferences or modify the Preference objects in any way. A Preference can be a Dorm Preference which indicates that the Preference deals with which dorms the User wants to live in. Dorm Preferences are handled differently than other Preferences in our suggestion algorithm. A Preference can also be a Roommate Number Preference which indicates the number of roommates the User would like to have. Roommate Number Preferences are also handled differently than other Preferences in the suggestion algorithm.
    - description: text description of the Preference. ex: I am ok with noise at night.
    - response: an opinion for the Preference. Can be one of: Yes, No, Don't Care. The default is Don't Care.
- **Group** is a list of Users that are roommates with each other. Users cannot be in more than one group. Users are added to a Group only if everyone in that Group approves them. Users can remove themselves from a Group at any time. Users cannot remove others from a Group. If there are less than two users in a Group, the remaining user is removed and the Group is deleted.
    - roommates: list of Users that have accepted to room with each other
- **Request** is a roommate request from a User to another User. When a User wants to be roommates with another User, a Request is made. The requested User can accept or deny the Request. The requesting User can cancel the Request before the requested User responds (by accepting or denying).
    - from: the User requesting
    - to: the User being requested
- ***Changes*****:** We added Group and Request to reduce data redundancy (before we had a list of roommates and a list of requests to/from instead). We've also added Dorm Preference as a special type of preference since it is treated differently. Moreover, we've changed the relationship between User and Preference: a user used to have their own list of preferences, but now they have a list of references to preferences, which are a unique combination of a description and a response). This also cut down on redundancy (there is now a set number of preferences, no matter the number of users).

Data Design



- **Reversal of relations** between Group/User because it is easier to deal with a group attribute than with a list of members. To add a user to a group and remove them from a group takes one step: changing their group, instead of searching and splicing a list. Also, searching for roommates is easy: just search for all users who have the same group reference.
- **Response** changed from more abstract, formal definitions of the possible responses to preferences to more user-friendly “Yes,” “No,” and “Don’t Care” because we felt these labels would be less confusing and more representative of typical human speech. In the implementation, the set relation is represented with an enum.

- **Preference’s isDormPreference** field was created as a boolean to aid in quick checking and filtering of preferences. This complemented the set relation, where every preference is either a Dorm Preference (isDormPreference is true) or not (isDormPreference is false).
- **Preference’s isRoommateNumberPreference** field was created as a boolean to aid in quick checking and filtering of preferences. This complemented the set relation, where every preference is either a Roommate Number Preference (isRoommateNumberPreference is true) or not (isRoommateNumberPreference is false).
- **User’s available** field was created as a boolean to aid in quick filtering in the suggestion algorithm. This also fit nicely with the set relation because there are exactly two states of this field (available/true, unavailable/false).
- ***Changes:*** Group and Request classes were added, isDormPreference and isRoommateNumberPreference were added, and the same preference changes were made as described under data model changes.

Security Policy

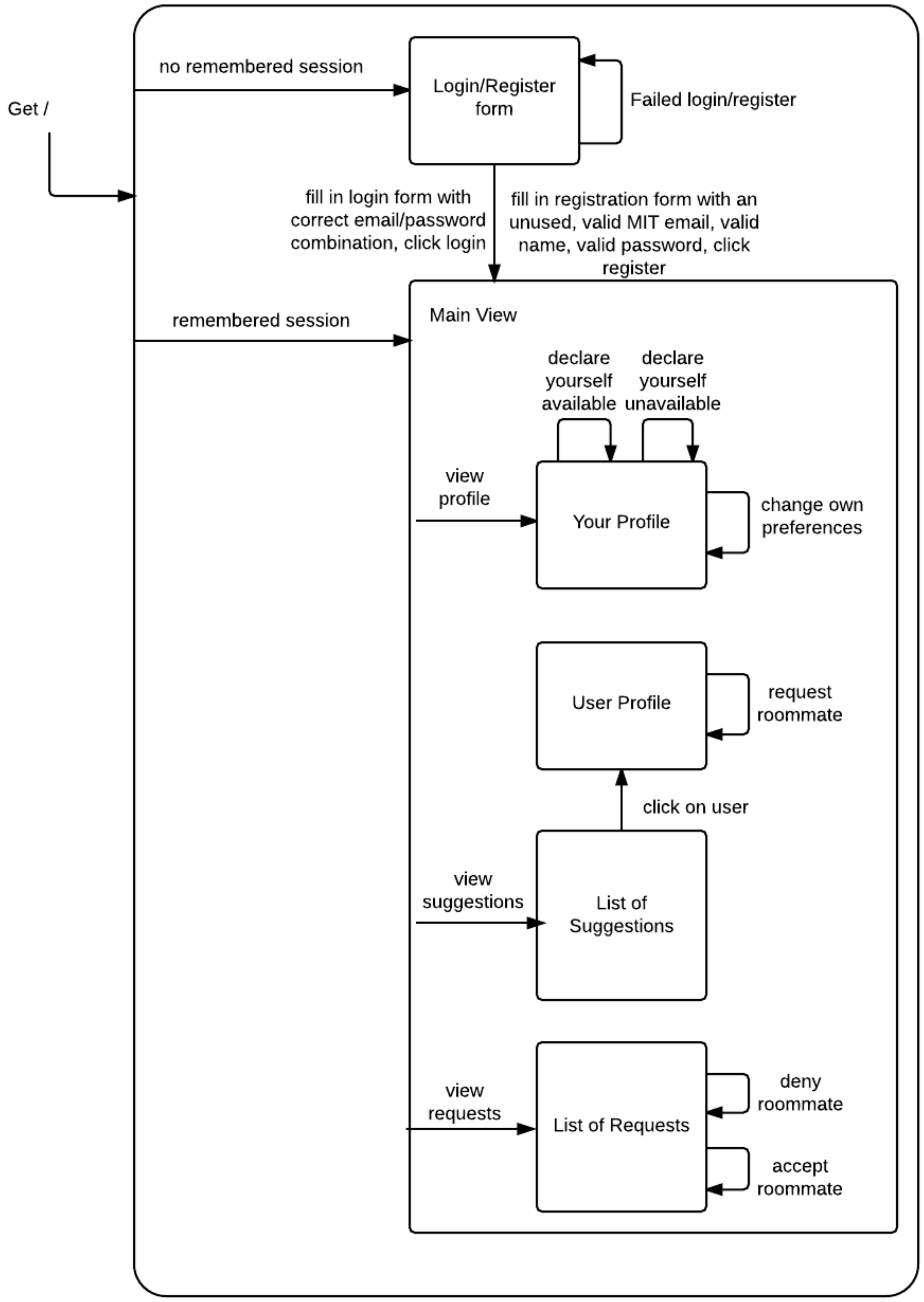
- Users have to login with a valid email and correct password.
- Users can only modify their own preferences/availability. Users can only accept/deny requests made to them. Users can only cancel requests made from them. Users can only dissolve the roommate group they’re in.
- Users can only see their matches.
- All users can see all other user profiles (and thus emails, roommates, availability, preferences).

Threat Model

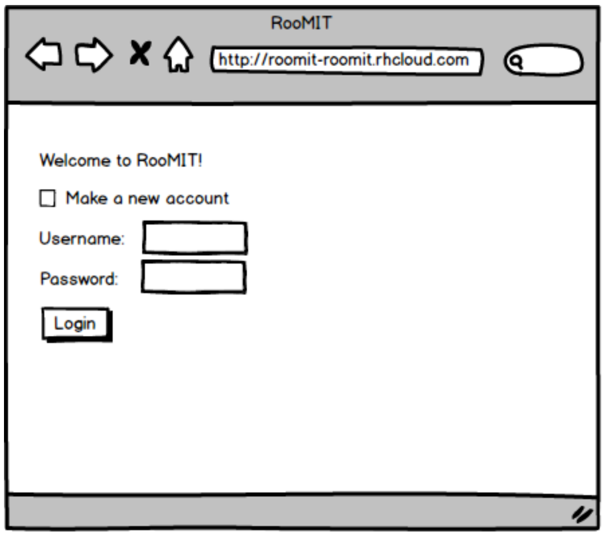
- Our app isn’t doing anything too crucial so we don’t anticipate any serious attacks.
- Attacker could gain MIT emails, but this is information anyone can access.
- Attacker cannot actually modify anything on MIT Housing via our site, so even if an intruder did manage to gain access, the only result would be that students are confused because the site isn’t functioning as expected.
- Attacker could try a MongoDB injection into our forms, but we validate both client-side (login/register: can’t submit incorrectly formatted inputs, preferences/availability: there are specific buttons to click, no user text input) and server-side (will not submit query unless correctly formatted and escaped).
- Authenticated attacker can construct requests (curl, Postman) and tamper with data. We will eventually create different levels of access for different users (eg. roommates can only modify each others’ availability and nothing else, users requesting other users can only modify the “requesters” attribute of the other user, etc).

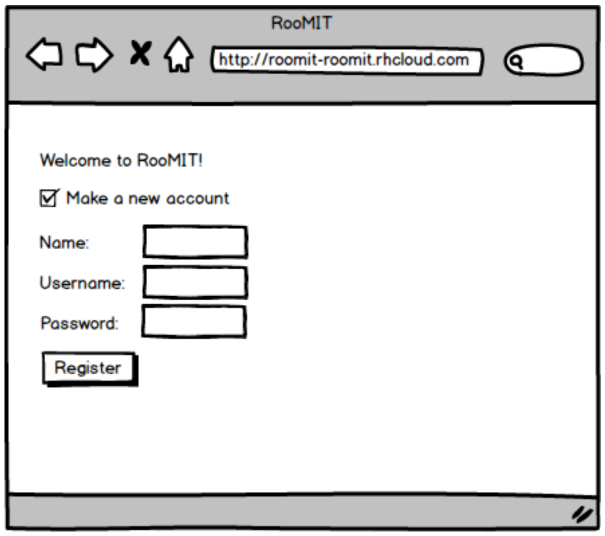
- Attacker could try an XSS attack, but once again, we sanitize our inputs and our requests are secure (no parameters passed through links).
- Attacker could try a CSRF attack, but we use csrf tokens (via the csurf module).
- Attacker could try to enter a non-MIT email, but we only accept @mit.edu emails (validation on client and server side).
- Attacker could enter bogus MIT emails. We will eventually mitigate this by sending confirmation emails.
- Attacker could create bogus MIT emails by creating mailing lists if the attacker is an MIT student (and could thus respond to the confirmation email). We don't view this as a very large security risk, as the MIT community is small enough that attackers are unlikely to be from MIT.
- ***Changes*****:** We've added in CSRF protection.

User Interface Wireframes

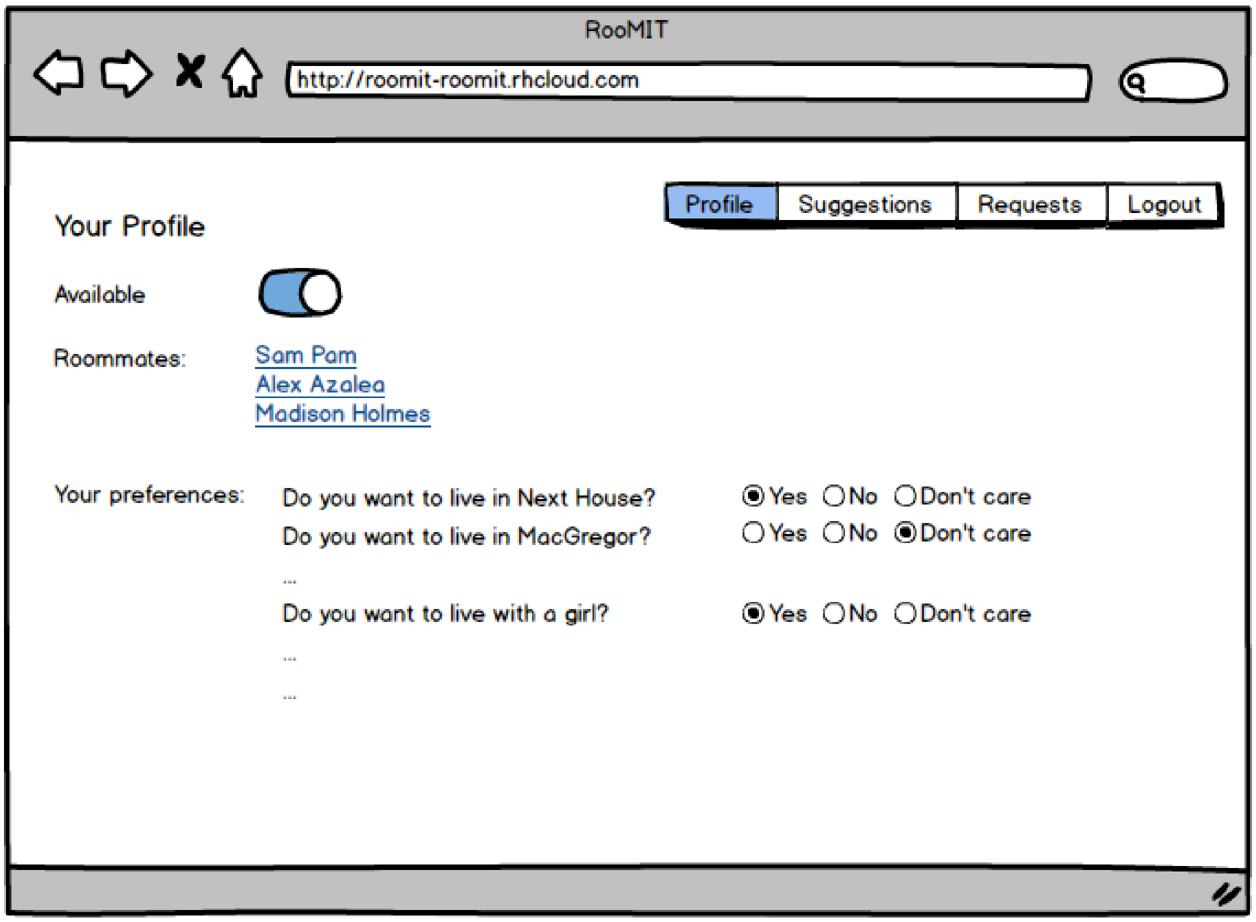


RooMIT is a one-page app. We have two views: the login/register view, and the main view. The main view will display the user's profile by default, and allows navigation to the list of suggested roommates and their profiles, as well as the user's list of pending requests.



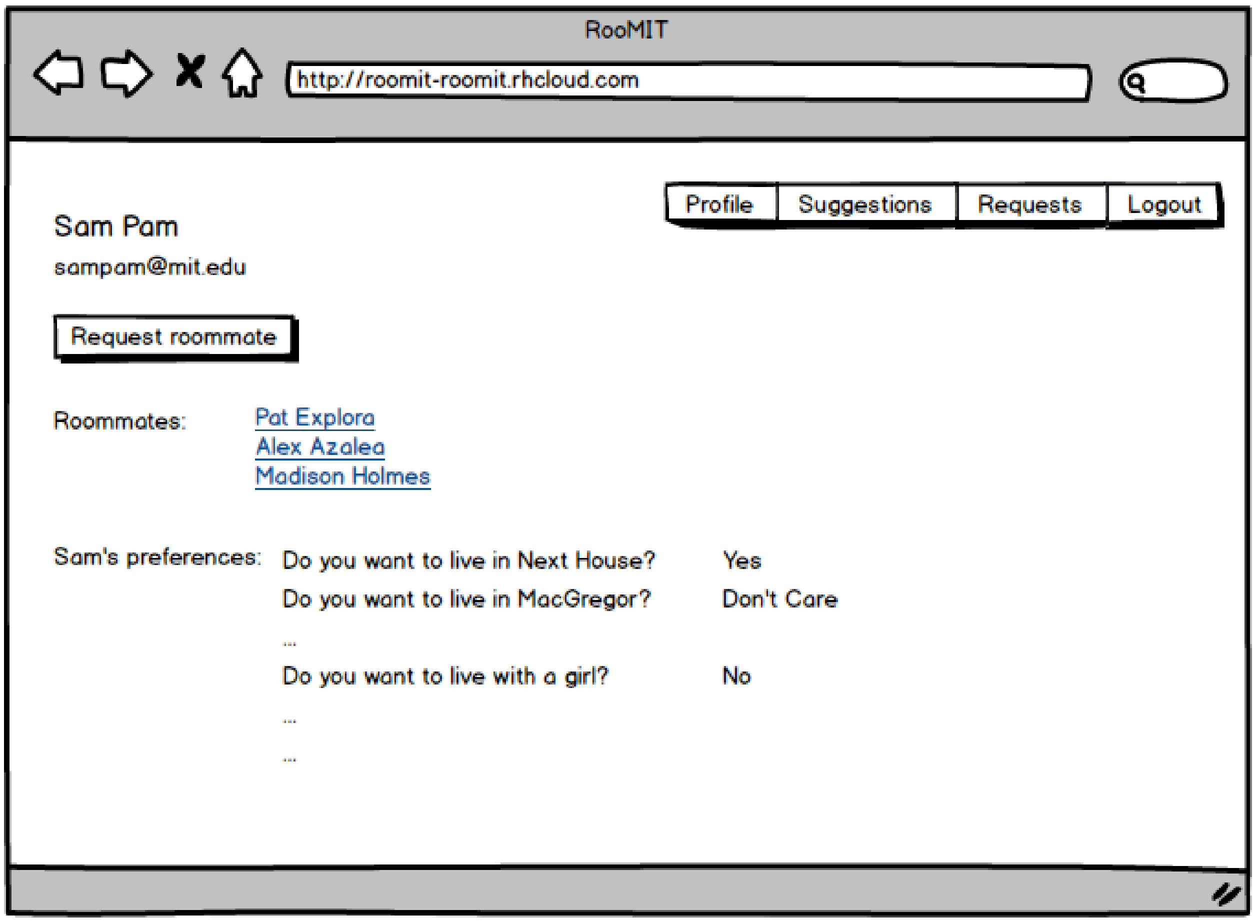


Login/Register page. Clicking the checkbox will toggle logging in/registering.



Do you want to live in Next House?	[✓ Yes] No Don't care
Do you want to live in MacGregor?	Yes No [✓ Don't care]
...	
Do you want to live with a girl?	[✓ Yes] No Don't care
...	
...	

The logged in user's profile. Here they can change preferences, view current roommates, and toggle whether or not they are available or not (searching for roommates).
Changes: Under the list of roommates is a button that removes you from the group. Yes/No/Don't Care radio buttons have been turned into buttons for easier clicking.



RooMIT

http://roomit-roomit.rhcloud.com

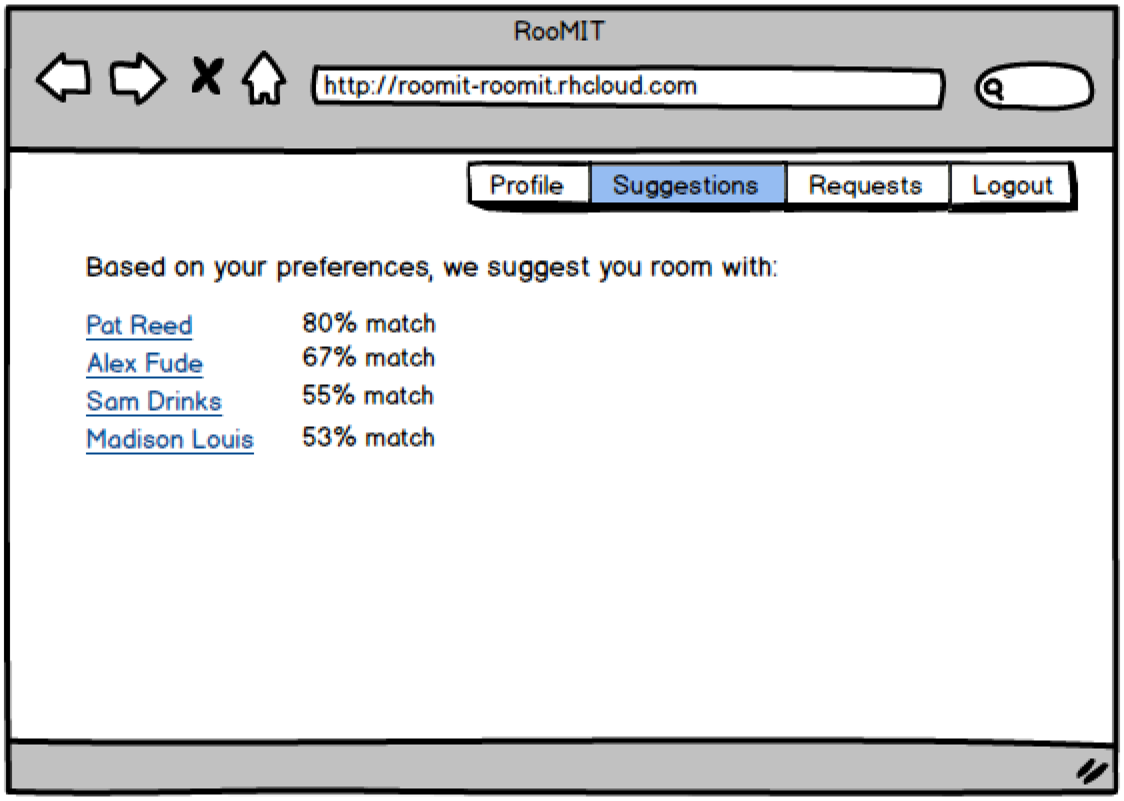
Sam Pam
sampam@mit.edu

Roommates: Pat Explora
Alex Azalea
Madison Holmes

Sam's preferences:

Do you want to live in Next House?	Yes
Do you want to live in MacGregor?	Don't Care
...	
Do you want to live with a girl?	No
...	
...	

User profile. Here their contact info and preferences can be viewed, and they can be requested as a roommate.
Changes: If the logged in user has already sent a request to the user whose profile they're viewing (here, Sam Pam), the "Request roommate" button turns into "Cancel request." If the user they're viewing (Sam Pam) has sent the logged in user a request, the button turns into two buttons: "Accept request" and "Deny request."

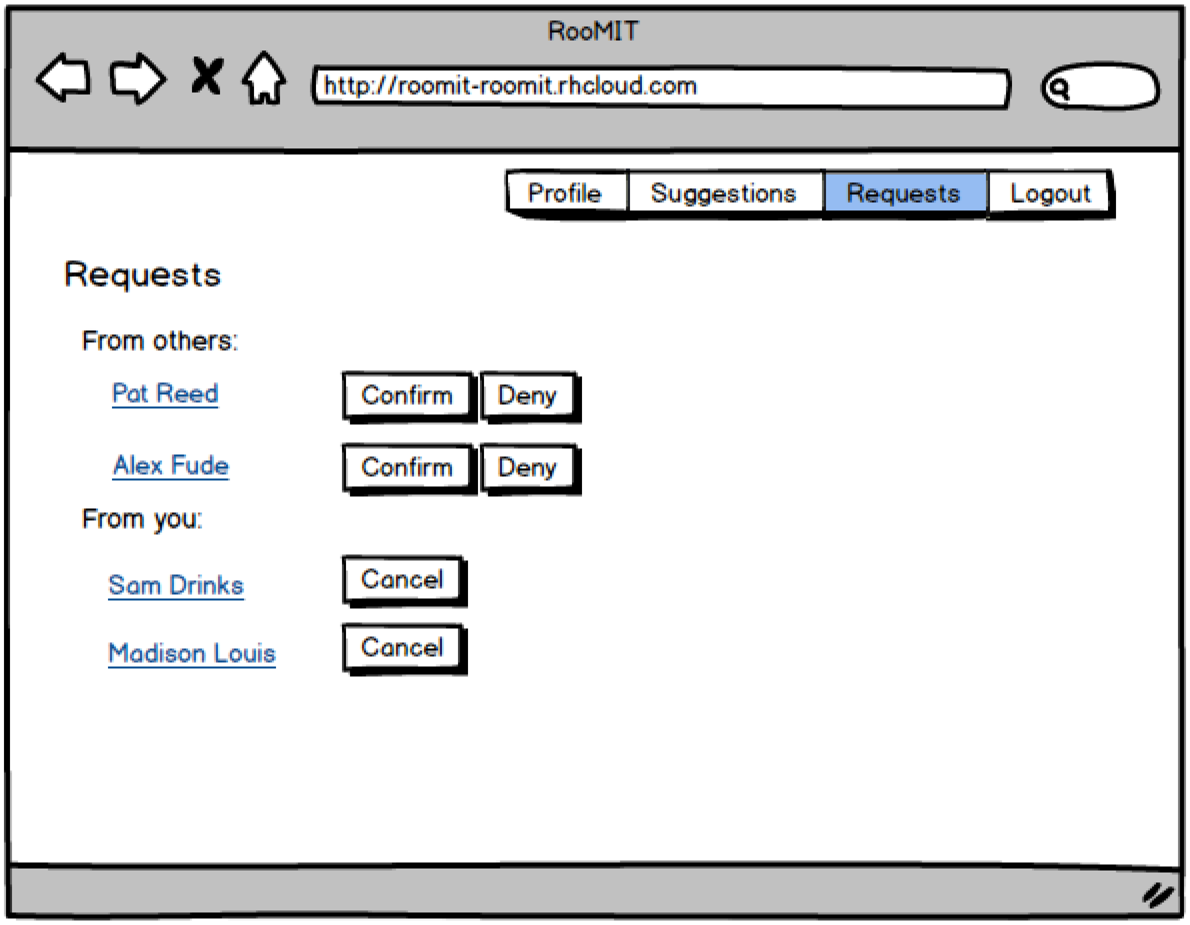


Based on your preferences, we suggest you room with:

Pat Reed	80% match
Alex Fude	67% match
Sam Drinks	55% match
Madison Louis	53% match

Suggestions page. This page displays potential roommate matches. By clicking on the users' names, their profiles can be viewed.

Changes: If users are roommates, they are grouped together. Also, emails are shown next to usernames.



Requests page. Here the user can approve/deny requests from other users, as well as view or cancel their own requests.

Design Challenges

- How do we represent Requests?
    - Options:
        - Have a list of requests from and to a user. This will require us to only run populate once when we retrieve a user object.
        - Have a Request class with a to and from field. This is more elegant and has less data duplication, but will require running populate twice.
    - Decision:
        - We did the first option for MVP, and the second for post-MVP to make the design more elegant and less redundant, and also to easier preserve the representation invariant (instead of two lists needing to be updated correctly, just one object with two fields needs to be created).
- How do we represent the User/Group relation?
    - Options:

        - Have each User have a Group. This option is more straightforward for editing.
        - Have each Group have a list of Users. This would require list manipulation when we need to edit a group.
    - Decision:
        - To avoid awful list parsing and splicing, and for easier user addition/removal from groups, we decided on the first option.
- How do we know how many roommates a user wants?
    - Options:
        - *Allow adding any number of roommates until the user indicates they are unavailable.* This option may result in suboptimal roommates who differ on how many roommates they would like to have.
        - *Have the user specify the number of roommates they would like.* Suggestions will only show compatible users that would like the same number of roommates.
    - Decision:
        - Have the user specify how many roommates they would like. This will help minimize any potential disagreement over the number of roommates to add between newly established roommates.
- How do we represent how many roommates a user wants?
    - Options:
        - Create a new field in user, specifying how many roommates the user would like to have. This option would make accessing this information easier, since we need it to suggest people and determine when a group is unavailable.
        - Include it in preferences. This option would simplify the design model, especially since this information can be categorized as a preference.
    - Decision:
        - Include it in preferences to keep our design model organized and the UI simple.
- How do we handle requests to a user with roommates?
    - Options:
        - *Send the request to the one user and have them approve it, and if they do, enter the user into the group*. This option is not ideal because this may be too much deciding power for the requested user, especially if they are in a triple or quad.
        - *Send request to all roommates individually and only add the user if everyone approves them*. This ensures that everyone is on board with adding a new roommate.
        - *Send request first to the user, then if they approve it send it to the user's roommates, and if they approve it, then add the user.* This may make the confirmation process unnecessarily long, since all roommates should have to confirm anyway for a new roommate to be added.

    - Decision:
        - We decided to send the request to all roommates immediately because we want everyone to approve the new addition, but don't want to complicate the process unnecessarily.
- How do we handle a user (A) with roommates requesting another user (B) who has roommates? (this can only happen with 2 pairs of roommates)
    - Options:
        - *Send requests from A to B and B's roommates*; if they all approve, remove A from A's group and put A into B's group. This will assume that A's roommate will make their own decision on whether or not to join B's group and will not necessarily want to stay with A.
        - *Send requests from A and A's roommate to B and B's roommate*. This assumes that A's roommate will want to join B's group if A wants to join B's group. This option may be hard to resolve because B and B's roommate may confirm different subsets of A and A's roommate.
    - Decision:
        - Send requests from A to B and B's roommates (first option). This is similar to how we handle requests if A did not have any roommates.
- How do we handle a user (A) with roommates requesting another user (B) who does not have any roommates? (user A can be in a double or triple)
    - Options:
        - *Send requests from A to B; if B confirms, then send request from B to A's roommates*. If A's roommates confirm then B joins A's group. If A's roommates deny then A has a choice to stay in current group or leave the group and become roommates with B.
        - *Send requests from A to B*; if B confirms, then A leaves A's group and becomes roommates with B.
    - Decision:
        - First option. What we do after B accepts is similar to how we handle requests of one person to a group of roommates.
- How do we show users who have roommates in the suggestion UI?
    - Options:
        - *Show them as individuals.* The user must go to the suggested user's profile to see their roommates, which will not be an additional step because the user must go to the user's profile page to request them. However, there will be more work for the user to see how well their suggested user's roommates match with them.
        - *Group roommates together on the page, but still only show users who are good matches with you*. This option ensures the user that only good matches are displayed on the suggestions page, but limits the user from finding out how compatible the roommates of a suggested user are with them.

        - *Show all suggested users on the page, and show the roommates of the suggested users and their compatibility*. This option allows the user to make sure they are okay both the suggested user and their roommates before making a request.
    - Decision:
        - We decided to show all suggested users and all their roommates because this would make it more clear and allow the user to see their compatibility with all roommates immediately.
- How do we handle roommate removal?
    - Options:
        - *Allow users to put in a request to delete their roommates (have every other roommate approve the change)*. This option would complicate our data model and may be redundant because each user in a group already has to confirm before a new roommate is added.
        - *Don't allow any removal.* This will ensure that roommates, once confirmed, are stable and will stay roommates. However, this allows for no room for change if someone changes their mind and wants a different living arrangement.
        - *Allow users to dissolve the entire group.* This option is simple to implement, but would give a lot of power to each individual user, especially if the group is 3 or 4 people wide.
        - *Allow users to only remove themselves from the group*. This option allows for flexibility in planning and does not require the confirmation of other roommates because leaving a group is a personal choice.
    - Decision:
        - Allow users to only remove themselves from the group. This would not overcomplicate the data model and will give each user the final say on whether to stay in the group.
- How do we represent roommates?
    - Options:
        - *Each user has a list of roommates*. This requires no extra classes, but requires manually editing each user's list every time there is a change.
        - *Create a Group class such that every user in a Group is roommates with each other*. Each user in the Group will have the same group id so that they can access the same group. However, making edits to add or remove people from a group requires only editing the Group, which is much easier especially in the context of having the option of more than one roommate.
    - Decision:
        - For MVP, we did the first option because users could only have 1 roommate. For post-MVP, we implemented the Group class option to make edits to roommateships easier.
- Should we incorporate dorms?

    - Options:
        - *Incorporate dorms.* It matters what dorm your roommate would like to live in, and it limits the pool of potential roommates to those who only want to live in the same dorm. Each dorm will be listed as a separate preference.
        - *Don't incorporate dorms*. Dorms are not central to our purpose and may add unnecessary complexity.
    - Decision:
        - We made the dorm you live in/would like to live in a preference, so we would only match people with similar dorm interests. This would prevent the app from suggesting roommates who in reality will not want to live with each other because of differences in dorm interests.
- What should the options be for responses to preferences?
    - Options:
        - *Make responses to preferences yes/no/don't care*. This option effectively categorizes user responses and helps simplify our suggestion algorithm.
        - *Make responses to preferences a scale from 1-5 and dont' care*. This allows for more nuance in user response, but will introduce more complexity.
    - Decision:
        - We made options yes/no/don't care to simplify the suggestion process.
- What should we use for login?
    - Options:
        - *MIT certificates*. This idea is difficult to implement but would limit the users to MIT students.
        - *Google login.* This would handle security issues, but it would not limit users to MIT students.
        - *Use our own system*. This would take some time and security management, but could limit to MIT.
    - Decision:
        - We decided on the last option because we don't want to allocate the majority of our work time to implementing MIT certificates, and we do want to limit the users to MIT students, so we can't use Google login.
- How many roommates should we allow?
    - Options:
        - *Only two*. Most dorms only have singles and doubles, and this is easier to implement.
        - *Unlimited*. There are doubles, singles, triples, and quads for now, but the system is ever-changing depending on rooming supply. This option is therefore more realistic.
    - Decision:

        - We decided to implement the latter because it doesn't add that much complexity, but improves the app's usability by a lot.
- How do we address the issue of gender?
    - Options:
        - Add a field to user for their gender orientation. The app will then only consider users of the same gender. This choice will abide by certain dorm rules that only people of the same gender can be roommates.
        - Add preferences for having roommates of each gender. This accommodates for the LGBT community and the move on campus to allow for mixed gender roommates (eg. Senior House).
    - Decision:
        - We decided on adding preferences for having roommates of each gender, because this allows for flexibility in case dorm policies on gender change in the future.
- ***Changes***: added discussion on how to represent requests and the user-group relation, and discussed how to deal with different roommate request, suggestion, or cancellation scenarios.