Design Document for MITspaces

By: Anthony Chen, Brodrick Childs, Nahom Workie

1. Overview

Searching for an empty room to work in is oftentimes a difficult problem to solve as a MIT student. A search or an empty location can bring your search through the Infinite and multiple floors of the student center. Our application MITspaces aims to solve this problems.

MIT spaces will be a Ruby on Rails application that displays possible study areas in the MIT community. The goal of our application is to provide a crowdsourcing platform to allow users to discover available rooms to study or work in.

User will be able to determine whether certain locations are currently occupied or not. After a location is chosen, the user will be able to check-in upon arrival and crowdsource the information about the room. This will allow other users of our service to know if the room is available or note. Similarly, users will check out when they are down with a particular room.

2. Concepts

The key concepts for MITspaces are Collaborators and Rooms. Collaborators use MITspaces to find rooms available to study in. The service connects collaborators to other who are available by connecting to their Facebook account. These Collaborations are able to check into Rooms, at which point MITspaces will track it's status.

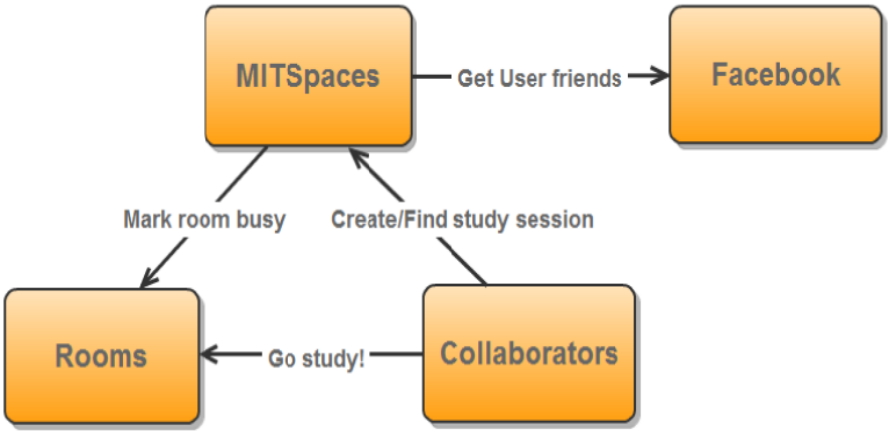

Figure 2.1: Context Diagram for MITspaces

3. Behavior

MITspaces Feature Description

1. **Secure Login** => Users can login to their accounts securely using their Facebook accounts. This enables the site to also get a list of the users friends and check if anyone of their Facebook friends also uses MITspaces.
2. **Find Room** => Users can find an empty room on campus. Using the Google Maps API users can see all the rooms on campus and their current availability. The map allows for an easy way to see exactly where a room is and when clicked users will be able to see more information about the room.
3. **Occupy Room** => Users can check into a room and have it marked as occupied for others to note. When a user checks into a room via mobile check-in the rooms occupation status immediately changes to true (i.e occupied). The color status of the pointer for the room on the campus map will also change to red to denote that it's been occupied. The map uses AJAX request to continuously update the current state of occupied and available rooms.
4. **Add Room** => Users can add a study room and have it approved by administrators. If a user has a room on campus that they study in and want to add it to the MITspaces database they can request to add a room. The user will input information such as Name, Location and Description. This request is then sent to the administrators who then have to check the room and approve it to be listed and shown on the MITspaces map.
5. **Delete Room (Mods)** => Administrators can delete rooms. Users can have a room to be deleted from MITspaces if students can't study in the room or if multiple users request that the room be deleted. Ultimately, this decision is made by the administrators who have the final word in the approval process.
6. **Find Friends** => Users can find their friends and if they have occupied a room. Using the Facebook API allows MITspaces to access a user's friend list. If the user's friends are occupying a room, when a user clicks on an occupied room MITspaces will show which friends are occupying the room when it displays more information about the room.

Object Model

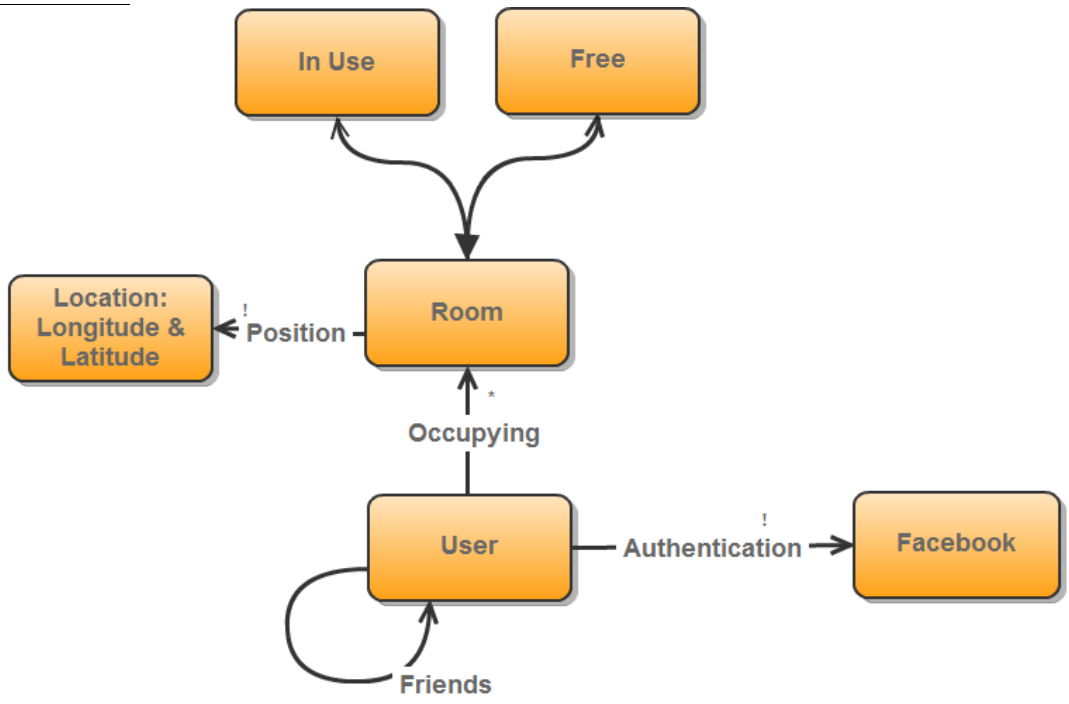

Fig 3.1: Object Model for MITspaces

Security

In this aspect of the project we handle security from a couple angles. Our largest concern is to make sure that our users are using the service in the manner in which it was intended. Since our application is built on the idea of crowdsourcing, we need make sure users are authenticated and they are submitting relevant information.

Login: Login is handled via Facebook, so we make sure the user is who they say they are that way. They will also be asked to confirm their MIT email address. Our goal with login is to be strict enough that we are only allowing users from the MIT community.

Cross site scripting/Access Control: The secure actions, like adding a room and checking out a room check to make sure the user has the proper authority and is able to make the action in the backend, so they are secured this way. If people try to send requests to the site with forged parameters to make or delete a note, it makes sure this user is logged in and has the proper credentials to perform the actions. Furthermore, the use of moderators adds an additional level of security in this aspect. Although the use of moderators increases the amount of time before a new room is submitted, it drastically improves the quality of information presented by our service.

User Interface

The figure below demonstrates the wireframe views for MITspaces and the flow between them. The focus of our interface is the main dashboard. On this screen, a user can locate where his or her friends are currently studying and which rooms are currently available or occupied.

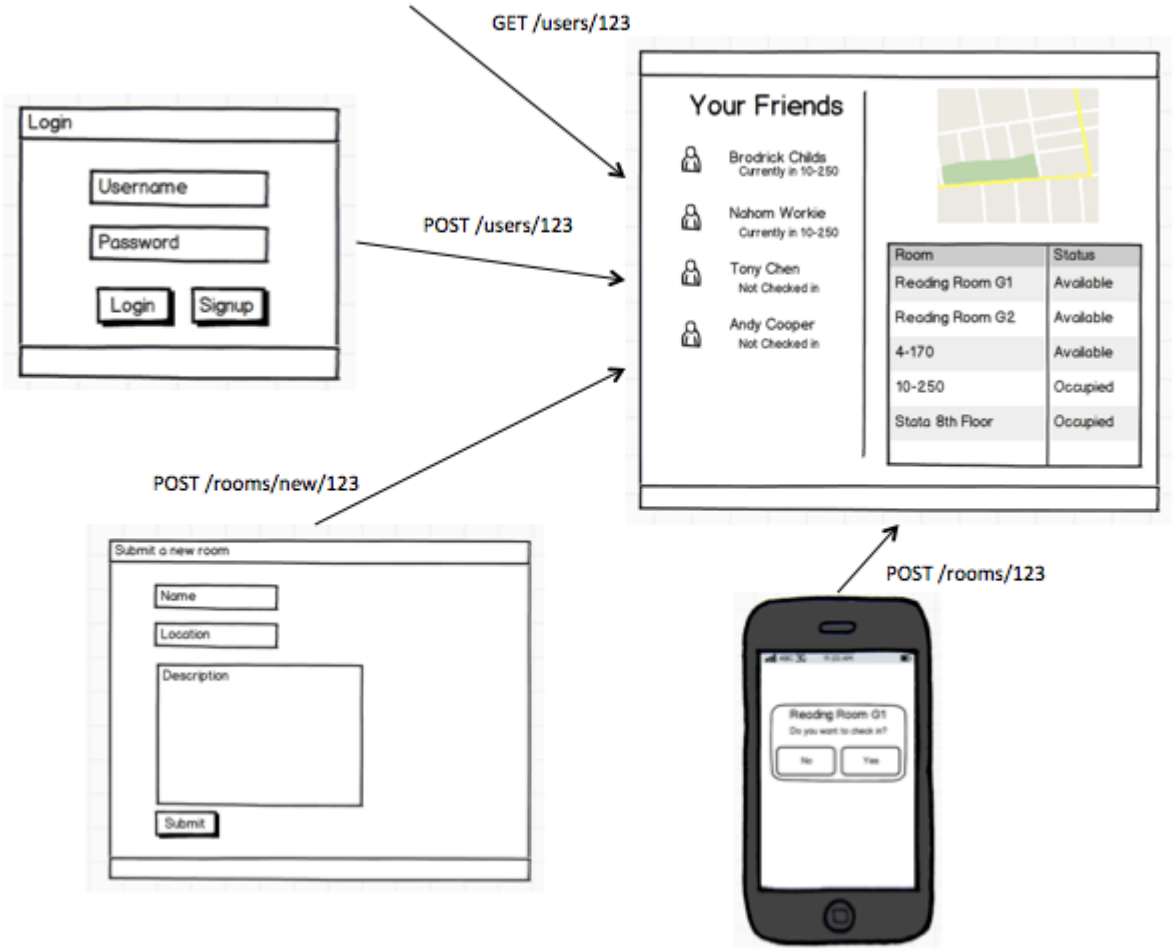

Figure 3.2: Wireframes for MITspaces

4. Challenges

One challenge was how to authenticate users. We want to make sure they're MIT students, so we need to send an email to an @mit.edu account to verify, but we also want the user's friends. Our options are listed below:

- Limited Authentication: utilize a rails gem and allow users to simply create an account with our application. This approach allows any user to create an account.
- Strict Authentication: force users to sign in with an mit.edu email address with veritification. This approach guarantees that we reach MIT students only, but requires a longer verification process.

We decided Facebook authentication was the best way to do this. Facebook will easily allow us to handle login and get a user's friends. We can easily send an email after the user has connected their account to facebook in order to authenticate them as an MIT student. Even

without MIT email authentication, we don't expect non-verified users to pose much of an issue since they can't do anything without physically being in the room to check out

Due to the importance of our users receiving the most updated information on the status of the rooms, we needed to decide the best approach to keeping them informed. Our choices are listed below:

- Constant ajax refreshes: continuously submit ajax requests to the server to maintain that the user has the most updated information. This approach would provide users with the most information, but it would require sending many requests.
- Manual refresh: Force the user to refresh the page if they want to see new information. This drastically reduces the number of requests made to our server, but limits the information they receive.

In order to improve the user experience, we plan on implementing an approach that refreshes the pertinent information in periodic time intervals. This way, the number of requests is reduced, but the users are still receiving updated information.

Another challenge is how to implement the room checking out.

- Best approach: Place QR codes or NF patches in the actual rooms, and force users to check out the rooms when they get there (and force them to check out when they leave.)
- Simple approach: If QR codes prove impractical, we could just have the users check out rooms by visiting the website.

We decided that we will talk with the MIT administration to determine whether or not QR codes are feasible, if not we will use the simple approach since it keeps the functionality within our site and has fewer external variables.