Project Name: Sweet Spots
Team Name: No REST for the Wicked
Team Members: Maryam Archie, Nishchal Bhandari, Bob Liang, Isaac Rosado

Project Pitch

https://drive.google.com/open?id=1KCw6sicKJC1R-BSKzIWvMqr6XtI0eiVBpxZbERStF0o

Overview

Brief Description of System to be Built

The system consists of a database of “Spots”, which are structures consisting of a title, location, a tag, and a list of reviews. Users can create, search for, and review spots through an interactive map based interface. Users will also be able to like or dislike other user’s reviews, affecting that user’s “Rep”, an indication of the quality of their contributions to the site. Besides the standard technologies taught in the class, we will be using React to handle the front end of the application and the Google Maps Javascript API to build an interactive map for the frontend.

Key Purpose

To find key points of interest on campus

It is not easy to find the “sweet spots” on campus without learning about them through word of mouth. Sweet Spots is a web application that makes it easier to find the little things that you need around campus. For example, you’re trying to find a room to work on your 6.170 project with your team but there are no available group spaces in the Student Center or Barker Library. By specifying what you’re looking for, the application will illustrate relevant points of interest on a map of the campus or simply in a list view.

Deficiencies of existing solutions

An example of an existing solution is the application Flushd, which allows users to find nearby bathrooms. The app even allows users to designate preferences such as a changing table, which would be useful for parents. The deficiency of the app is that there are no overall quality reviews of the bathrooms designated. Also, the application applies at a larger scale than just MIT, so the nearest bathroom shown by the application may be farther than ideal.
The website whereis.mit.edu is similar to the type of application we’re creating, but it is designed around the purpose of finding locations of rooms, restaurants, and other large spaces. Our focus is on finding more particular categories of things, such as water fountains and places to nap. Additionally, whereis.mit.edu lacks the functionality of allowing users to create and rate locations, which is a key part of our application.

Concepts

In our application, there are two main concepts: Spot and Rep.

Spot

Description
A Spot encapsulates the information about the point of interest - its name, location, category and the reviews associated with it.

Purpose
The purpose of a Spot is to find points of interest that are nearby and high quality.

Operational Principles
A user can create a Spot by dragging a pin onto the map which defines the location of the Spot. Afterwards, the user will input its name and then select a tag to categorize the Spot. The user can select *one* of the predefined or user-defined tags or create a tag of their own. The user must then review the spot, giving a description and rating (out of 5) of the Spot. Once the user clicks done, they will not be able to edit the properties of the Spot. Other users will be able to like or dislike the review about the Spot. Only users that are logged-in may create a Spot.

When the user would like to look for Spots, they must perform a query containing the name of the Spot or a tag and/ or a location. Once the query is complete, the Spots that satisfy the constraints will be presented on a map and in a list. The user may also choose to filter spots based on quality, that is the average rating of a Spot.

A user may favorite a Spot to reference it later.

To learn more about the Spot, the user can hover over the Spot and/ or click the Spot. At this point, the user can like or dislike the review associated with the Spot if they are logged in.

Anticipated Misfit(s)
See Challenges -> Design Risks -> Anticipated Misfits

Rep

Description
Rep is a measure of how useful a user's reviews are as well as how popular the Spots they submitted are. It is the sum of the scores of the reviews posted by a user and the number of reviews posted for Spots that the user created.

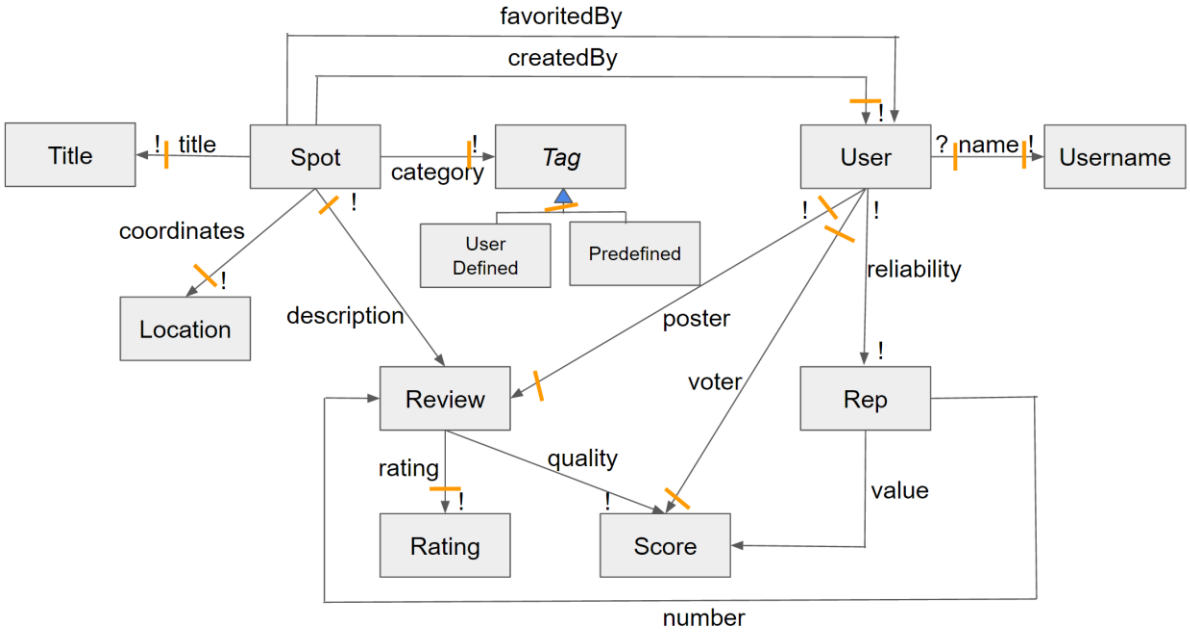
Purpose
To reward users for submitting Spots and reviews.

Operational Principles
A user can like or dislike a review for a Spot, which increments or decrements the score by 1 respectively for the review. The reviewer's Rep is adjusted accordingly.

If a registered user reviews a Spot, he/ she boosts the Rep of the user who posted the Spot.

Anticipated Misfit(s)
See Challenges -> Design Risks -> Anticipated Misfits

Data Model



(Textual constraints on next page)
Textual Constraints

- A User cannot score their own Review
- A User can only post one Review for a Spot
- A User's Rep depends on the scores of the Reviews it created and the number of Reviews their Spots receive
- A User can only upvote a Review once
- A User can favorite a Spot once
- A User cannot favorite the Spot they created
- User can only delete their own Spot during the first day that the Spot is created

Explanations

- The title of a Spot is the name of the point of interest the Spot represents.

- Based on the Reviews of a Spot and the Rep of the User who posted the Spot, a User can decide whether a Spot is indeed a “Sweet Spot”.
- The geographical coordinates of the general location define the location of a Spot.
- The score of a Review is the difference between the number of likes and dislikes.
- The value of a User’s Rep is equal to the sum of the scores of that User’s Reviews and the total number of Reviews across all Spots that the User has submitted.

$$Rep = \sum_{i=1}^{n} NumReviews(Spot_n) + \sum_{i=1}^{m} \big(Score(Review_m)\big)$$

$$= \sum_{i=1}^{n} NumReviews(Spot_n) + \sum_{i=1}^{m} \big(NumLikes(Review_m) - NumDislikes(Review_m)\big)$$

  Where n is the number of Spots the User has created and m is the number of Reviews that the same User has created
- Example: User X has created Spots A and B. A received 40 Reviews from other Users, and B received 10 Reviews. Additionally, User X has submitted one Review, which received 10 likes and 20 dislikes (meaning that the Review has a score of 10-20=-10). The User’s Rep would be 40 + 10 + (10-20) = 40.
- Note that the rating of a Review does not affect a User’s Rep.
- The “voter” relation between a User and a score means that a User affects the score of a Review when the User votes on that Review (thereby becoming a “voter”.)

Insights

- Spots can change, such as a bathroom being closed, which would then cause a previously correct Spot Review of that bathroom to now be false. Our application does not take this sort of change into account because the rate at which MIT changes its infrastructure is slow enough such that these occurrences can be dealt with as outliers.
- A Spot can be useful for more than one function. Thus, more than one Spot must be created for the point of interest. For example, if a place is both a good study space and a good napping place, a User may create two different Spots.
- The score for a Review may be negative.
- A User may have negative Rep.
- A User may search for Spots near a room. However, we only support building names and numbers when querying location.
- Users do not know who posted a Spot.
- Although a Spot may have many Reviews, it may not be a Sweet Spot. That is, the Reviews may be negative. Thus, queries may return both “Sweet” and “Bitter” Spots, which may clutter the User’s view.

Security Concerns

Key security requirements

The main security requirement that our application will utilize is authentication. Each user must be unique because each user needs their own rep, so we must have a method of authenticating users with their profile such that each user can only access their own profile. Also, only users that are signed in will be authorized to create and review spots. We will be using the bcrypt node module to encrypt a user's credentials by salting and hashing them. Thus, it becomes increasingly difficult for an attacker to retrieve a user's credentials. We also do not want people to obtain a user's credentials, so we will need to implement a system of automatically logging the user out for inactivity.

Standard web attack mitigation

We will not use eval anywhere in our code. We will also implement our routes such that request calls must come from appropriate referrers only. We will also not expose our rep throughout the code such that users cannot access our created object. Since our application uses React, and React takes automatic sanitization measures during the rendering process, we can prevent possible XSS attacks. If we take the example from class where an attacker posts a script that sends the user to the attacker's website, React will convert everything in the <script> tag into a string before the post is run, which means that the attacker's script will not execute. This process will also prevent injection attacks since the attacker's script will not affect the server. To prevent CSRF attacks, we will ensure that our get requests never make any changes to the database. We believe that it is unnecessary to take drastic measures to prevent CSRF and we acknowledge that our Spots database may be susceptible to attack. Given the time constraint, we believe that omitting the use of double tokens and referrer headers are within the scope of our project's goals. Lastly, we will be using Mongoose as our database which prevents injection with fixed calls.

Threat model

User

The user would likely not want to be constantly logged in and out as they would want to quickly open the application and find nearby points of interest. To both reduce attacks on the user's credentials and reduce login times, the default of the application will be to not login, but rather view the app as a guest, where the guest is given a read-only view of the spots and the information associated with the spots. Only when a user wants to review or add a spot will logging in be required.

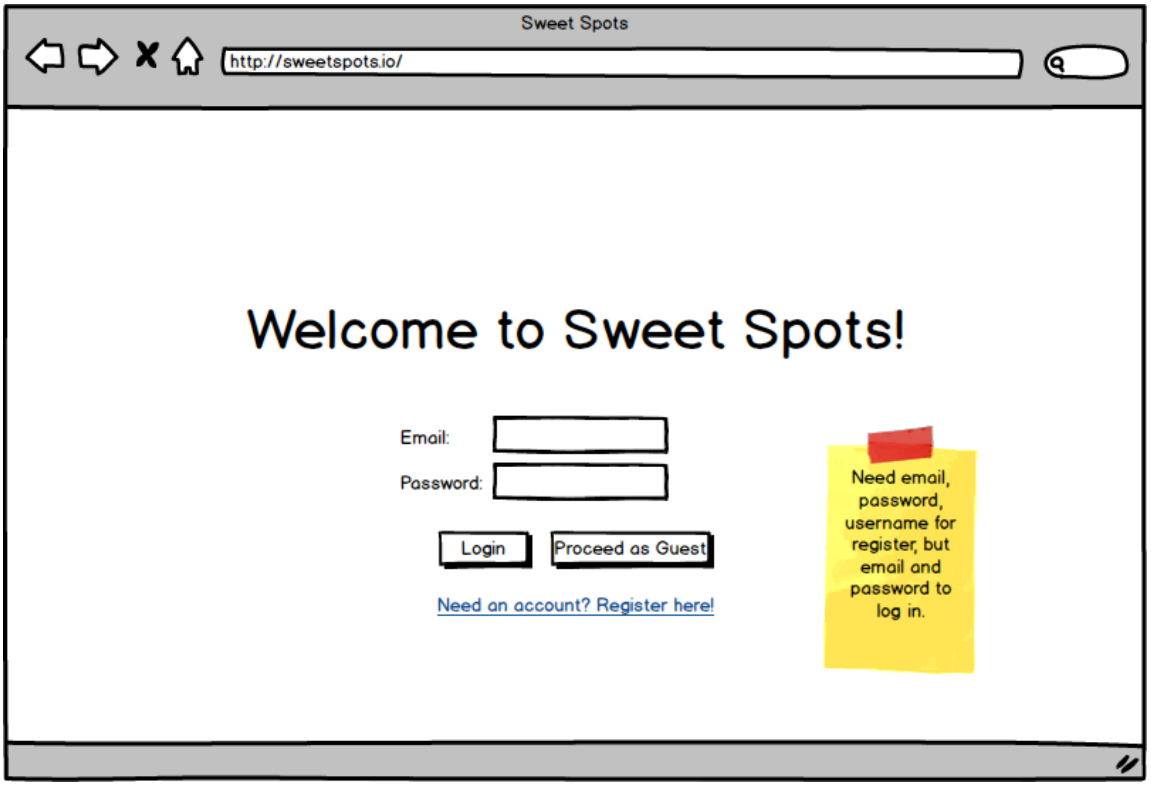
Attacker

The information that an attacker will want to obtain most is the user's authentication credentials. Obtaining the user's authentication will allow an attacker to access sensitive information such as student accounts. To prevent attackers from obtaining this information we will minimize the number of times a user will need to log in, and enforce automatic logouts.

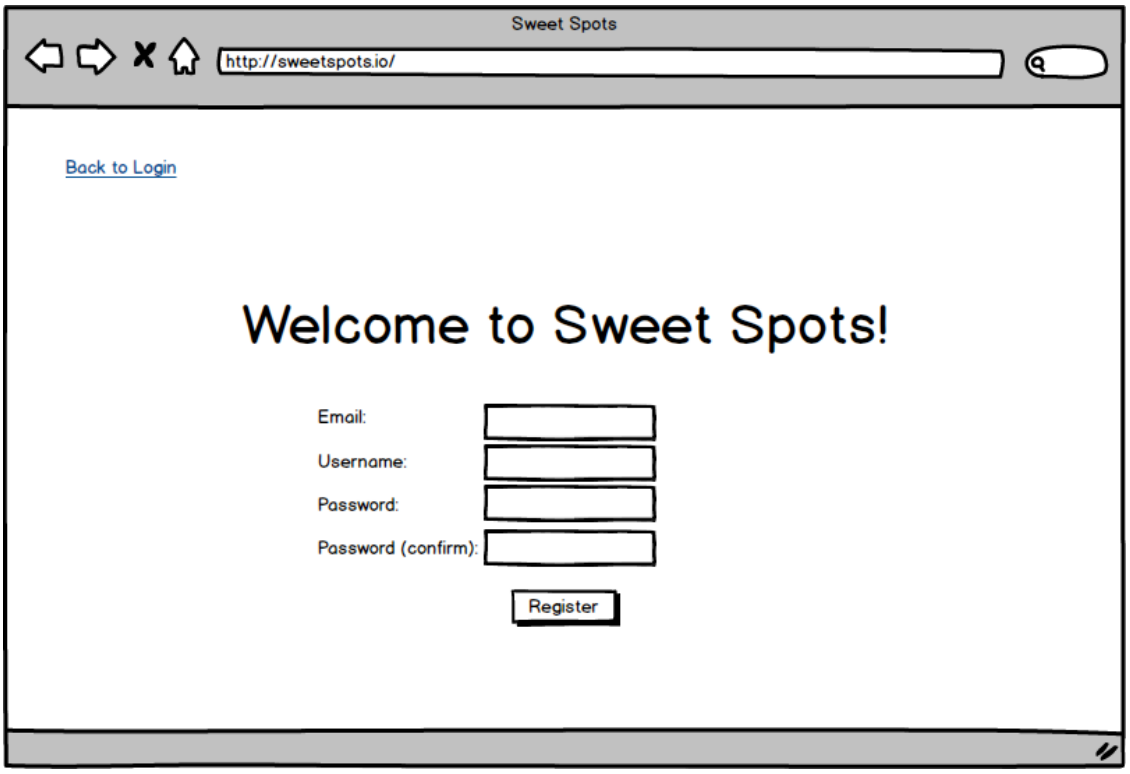
User Interface

Wireframes

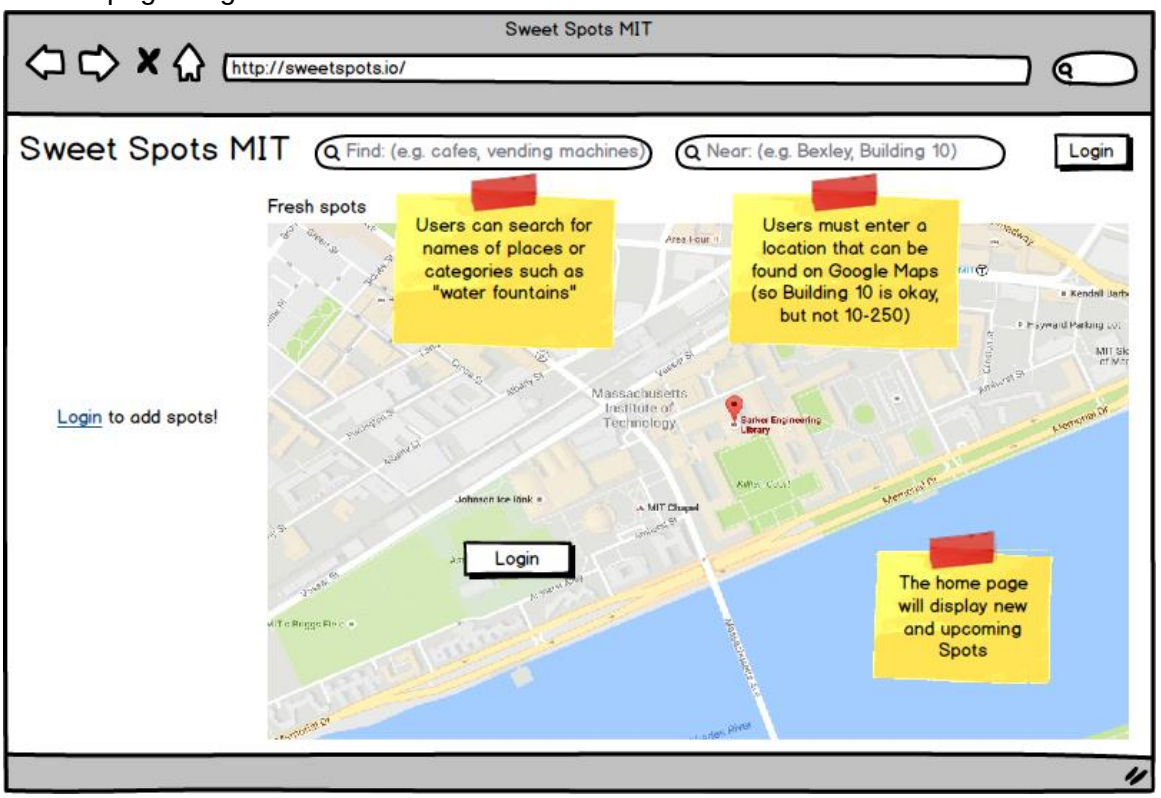
1. Login page



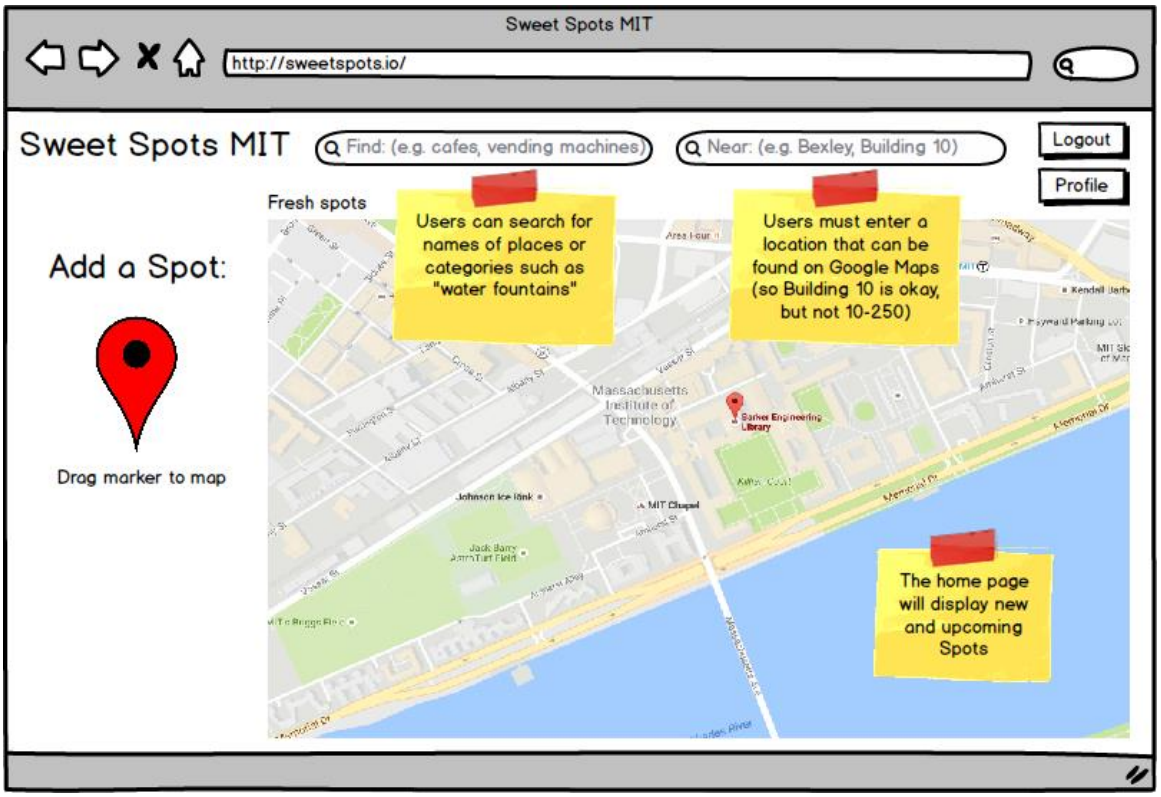
2. Register page



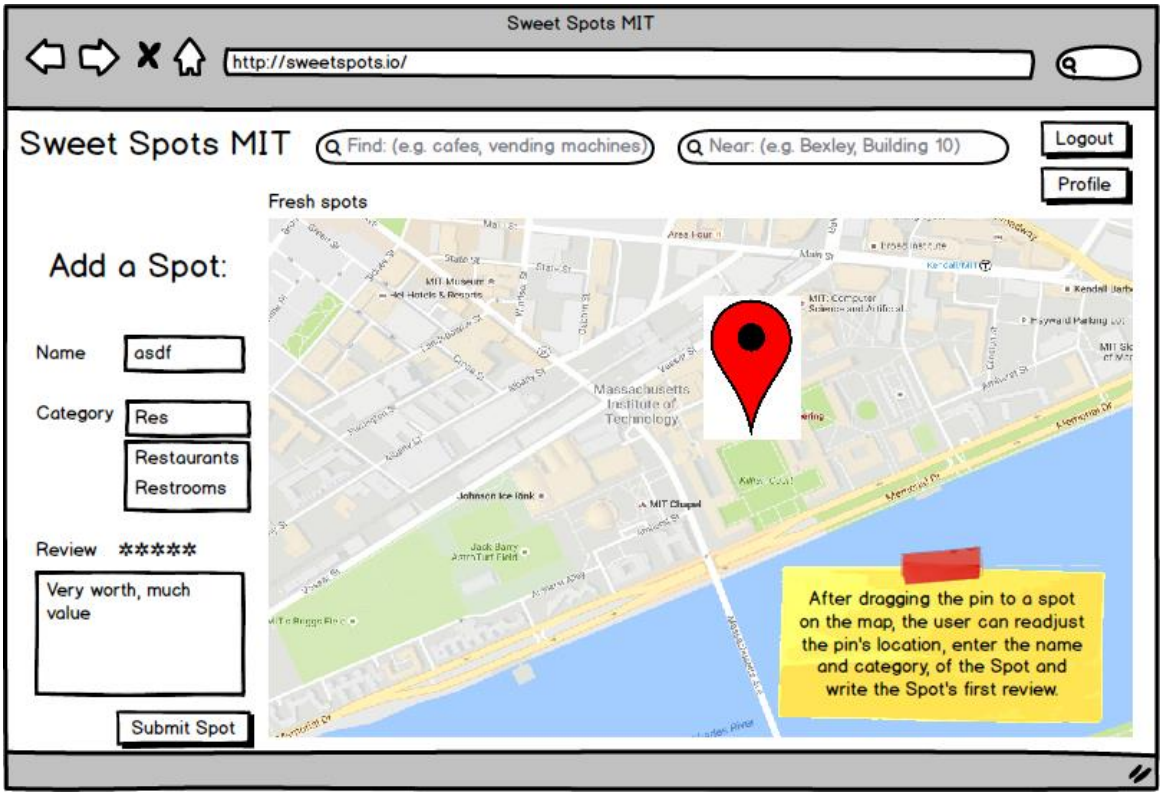
3. Main page as guest



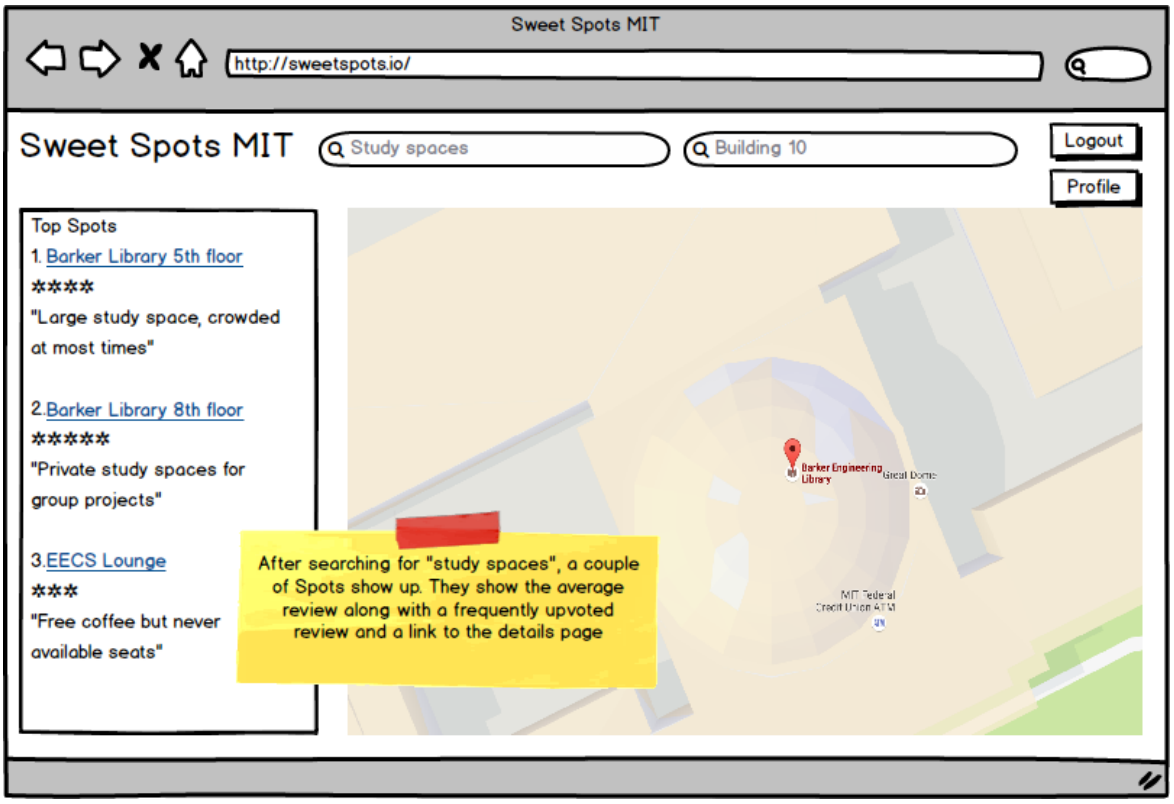
4. Main page logged in (Add Spot page)



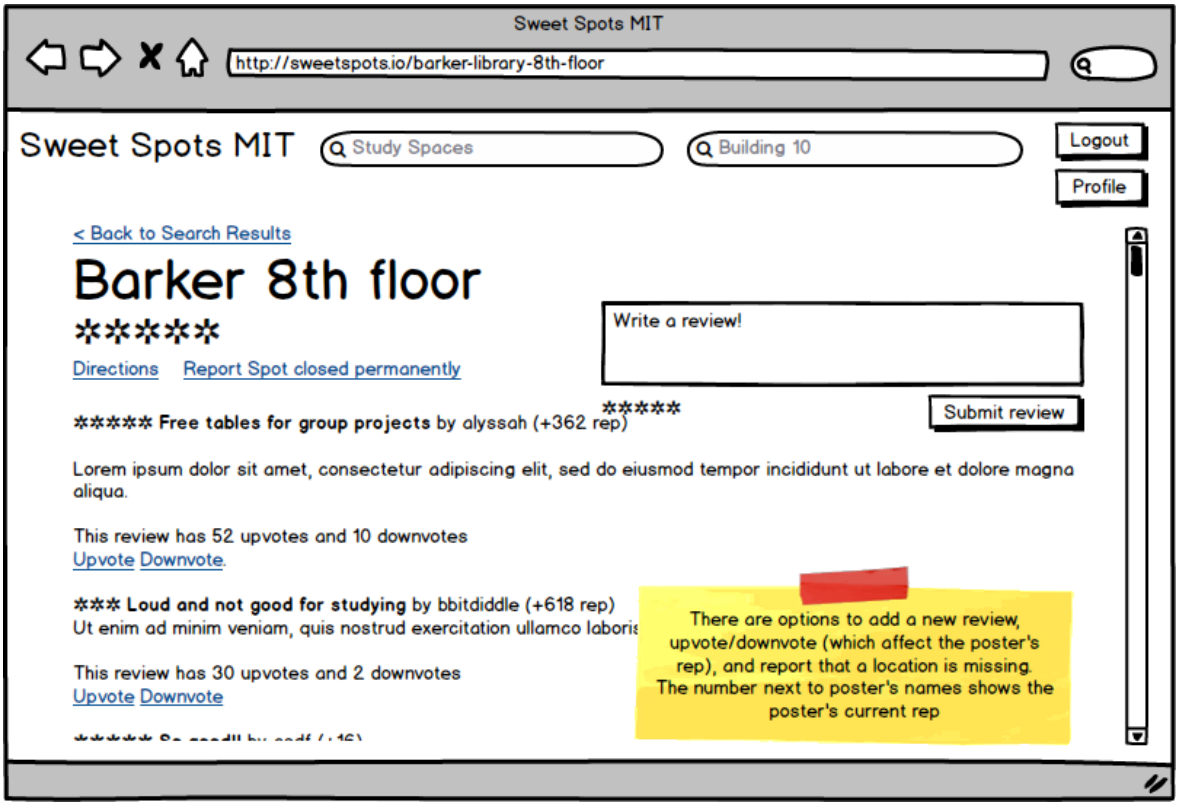
5. Add a spot page



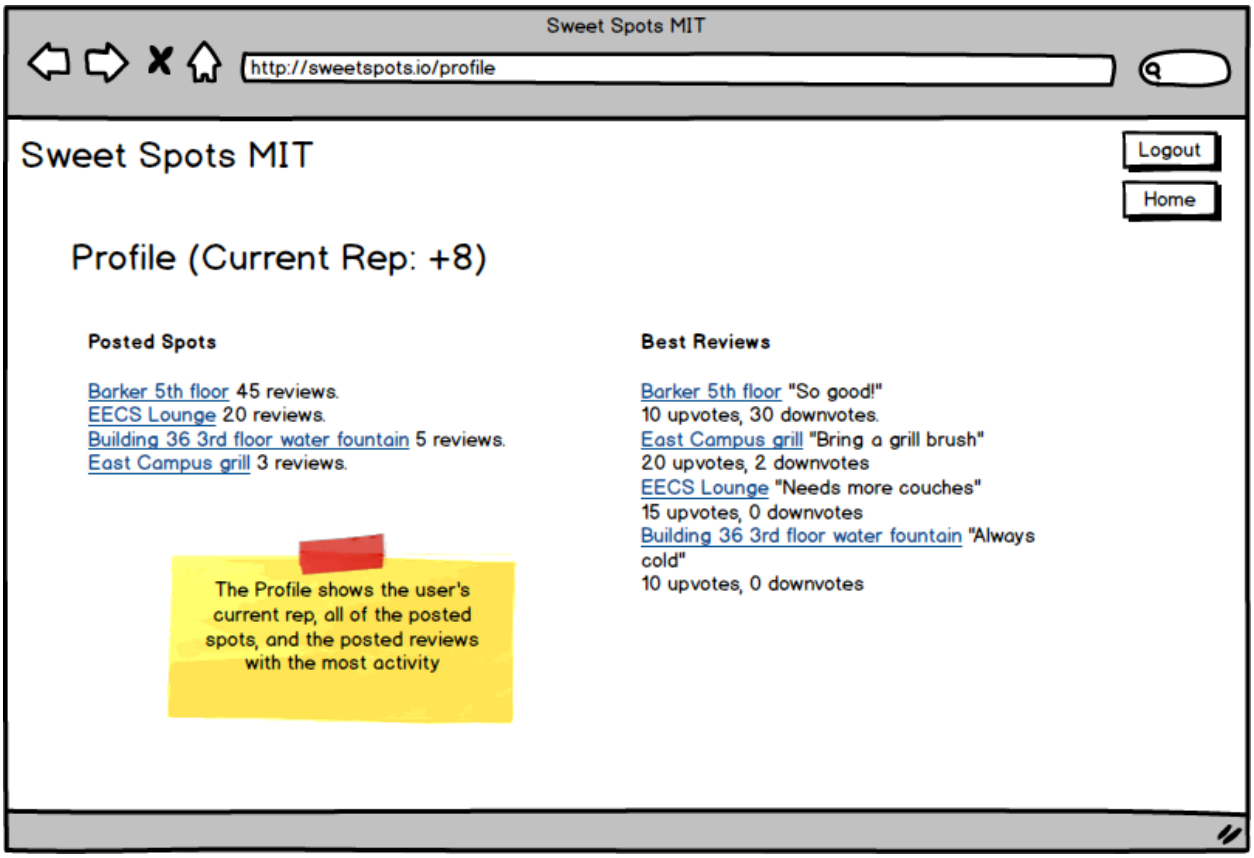
6. Search results page



7. Details page

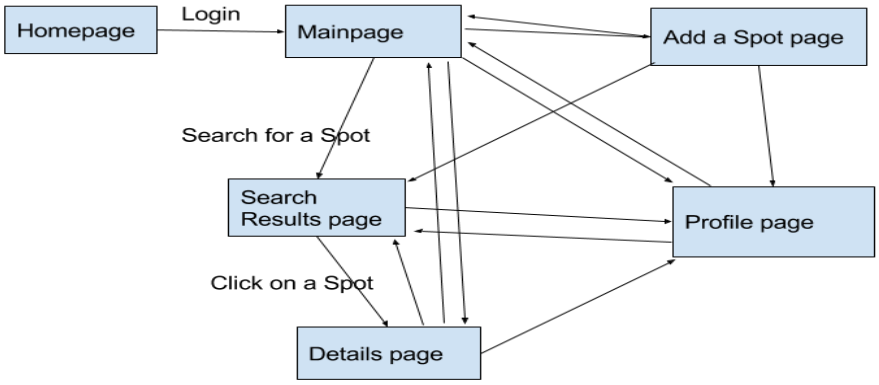


8. Profile page

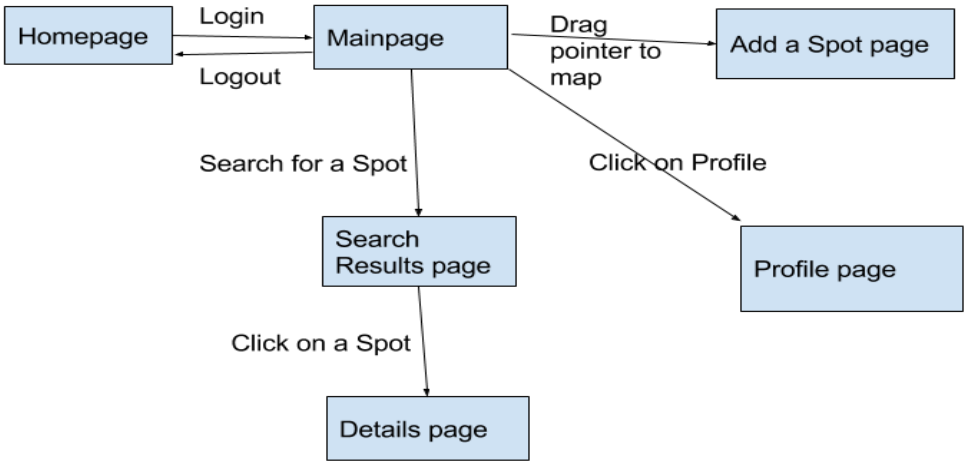


Transition Diagrams

Full Map (not showing arrows pointing from every page to homepage)



General flow:



- Clicking on Logout anywhere leads to homepage.
- Searching a location from anywhere leads to details page.
- Clicking profile anywhere leads to Profile page.
- Clicking on a location on the map in the main page leads to details page.

Challenges

Design Risks

Anticipated Misfits

For Spots, one anticipated misfit is that a spot is not easy to find for a user, or that the spots that a user gets for a given criteria do not match what they were looking for. For example, a user searches for a certain type of spot but the results they get are high ranked spots in another category.

Another anticipated misfit is that a spot should presently exist. For example, in 2012 a user could have added “Bexley” as a study space. However, in 2016, when a user searches for study spaces it would be a bad choice to return “Bexley” as a good study space. Conversely, another misfit could be that the user is searching for a “bathroom” near them, but the application says there are none near them, even though some do exist but have not been submitted by any other users.

Mitigations

To ensure that a user can find the spots that they are looking for, we must design the front end of the application in such a way that the user can define their search in such a way that gives enough information to the application for it to return desirable results. To do this, we should use our domain knowledge, as MIT students, of the kinds of things that MIT students tend to want to find.

To handle spots that are no longer valid, we will implement a “Report Invalid Spot” feature. In addition to being able to handle no longer valid spots, this feature could handle false submissions from malicious users. Based on a function of the reps of the users that report a spot as invalid as well as the overall popularity of a spot based on number of reviews, our system will automatically remove a spot once a certain threshold is reached. The benefit of taking into consideration user rep and spot popularity is that it will make it difficult for malicious users to get valid spots removed from the system. There is an additional benefit that user rep becomes more desirable, encouraging users to submit spots and reviews.

Finally, to ensure that users can find good spots, we should encourage users to submit new spots as well as reviews of existing spots. We will implement a user “rep” system (like the concept of karma on Reddit) to accomplish this. A user’s rep will be influenced by the scores of the reviews that they write as well as the popularity of the

Design Choices

Users submitting incorrect spots

A problem that may occur is when a user places a spot on the map and gives the spot an incorrect tag. Once this happens, we will need some sort of method of removing the spot. Some different options discussed include allowing for users to report and remove illegitimate spots, or having some sort of automatic system for removing illegitimate spots. If users can delete spots this will make them feel more empowered with the application, but it also allows users to report and remove correct spots. The advantage of the automatic system is that spots cannot be arbitrarily removed, but the automatic system may take longer to get rid of illegitimate spots. We have chosen to implement a system where a spot will be removed when the sum of the reps of the users that have reported the spot surpasses a certain threshold, which incorporates both automatic and user controlled removal of spots.

Handling duplicate spots

If one or more users submit spots that are essentially the same, we need some sort of way of reducing the spots such that only one accurate spot persists. Options include keeping only the first spot submitted, keeping the last spot submitted, or consolidating the two spots to form a meta spot. The advantage of keeping only one of the spots is that the implementation will be easier, but the spot kept may receive a lower rating than the other spot would have received. Therefore, we will implement the system that consolidates the spots, which will incorporate both users' reviews.

App view

Sweet spots could function with either a map view or a list view for the UI. The map view has the advantage that it immediately provides the user information related to the location of the spot, whereas the list view gives the user an indication of the priority of different spots. Since users will likely want both views at different times, we will implement a UI that gives the user the map view by default so that users can browse and add spots. Once a user searches for tags or clicks on a spot, the list view will be provided so that users can receive spots that match the user's query and in a specific order.

Tags

A decision related to tags is whether the tags will already be decided by the app or whether users will submit their own tags. If tags are decided upon beforehand, then adding spots will be easier as the user will not need to create a tag. Unfortunately, users may not have access to tags that they would find useful. If tags are not decided upon beforehand, then users must create tags every time they want to add a spot, which users would likely find annoying. We have decided to utilize a combination of these two, where a certain amount of pre-defined tags will be available for users to use, but users will have the option to add custom tags if they so desire.