Foodback MIT

Design Document - Phase 2

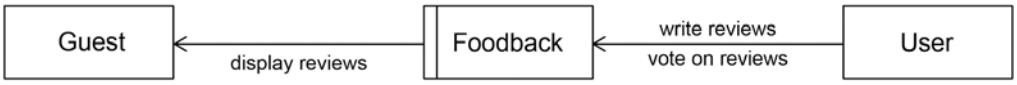
Motivation

Foodback MIT (or just "Foodback" within the MIT community) is a web application that allows members of the MIT community to post and read reviews of the MIT dining halls. Users must be logged into accounts created with @mit.edu email addresses in order to post reviews, but the posted reviews are fully public.

The purposes of Foodback include:

1. **Help MIT meal plan subscribers decide where to eat.** By reading reviews from other members of the MIT community, meal plan subscribers can better evaluate their options and are better able to decide which dining hall they would most like to go to. Foodback will allow users to read reviews on a per-dining-hall and per-meal-period basis, facilitating informed decisions.
2. **Help guests to MIT decide where to eat.** Guests generally have little to no knowledge of the MIT dining halls. By reading the reviews for each dining hall before their visit, guests will be able to determine which dining hall best suits their tastes.
3. **Allow MIT dining hall staff to see how they're doing.** Since the Foodback website is open to the public, the staff of each dining hall can read their respective reviews and determine what they're doing well and what they can improve.
4. **Allow MIT students to voice their opinions on the dining halls.** Currently, students can provide feedback to the dining halls via paper comment cards or an online form. Foodback seeks to improve upon both of these methods. The website will allow students to leave feedback from any location, unlike the comment cards which are physically located in the dining halls. The website will also have a more user-friendly and intuitive UI than the official dining feedback form, and the submitted reviews will be public, rather than disappearing into the "black hole" of the current feedback submission box.

Context Diagram



Concepts

Dining Hall: The specific MIT dining hall for which a review is submitted. Dining halls are identified by the name of the dorm in which they're located. Users can sort reviews by dining hall.

Meal Period: A period of time during which a dining hall offers a certain meal (breakfast, brunch, lunch, dinner, late night) and therefore has a certain menu. The available meal periods depend on the dining hall (only Maseeh offers lunch and only Simmons offers late night) and the timespan covered will also

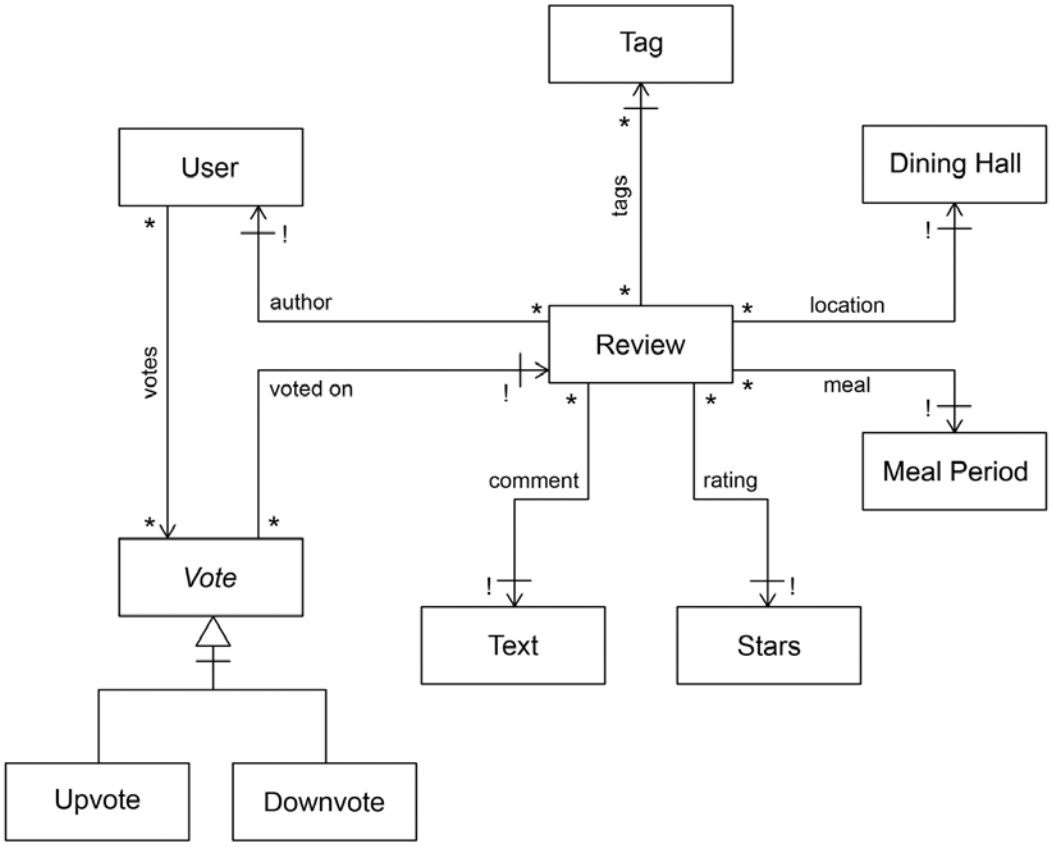
depend on the schedules of particular dining halls. Meal periods are also identified by name. Users can sort reviews by each meal period at each dining hall.

Review: A piece of feedback submitted by a member of the MIT community. Reviews consist of star ratings and textual comments, and are given for a specific dining hall and meal period. Unlike the physical comment cards that are posted on the wall in dining halls, reviews are meant to be more substantial in their content.

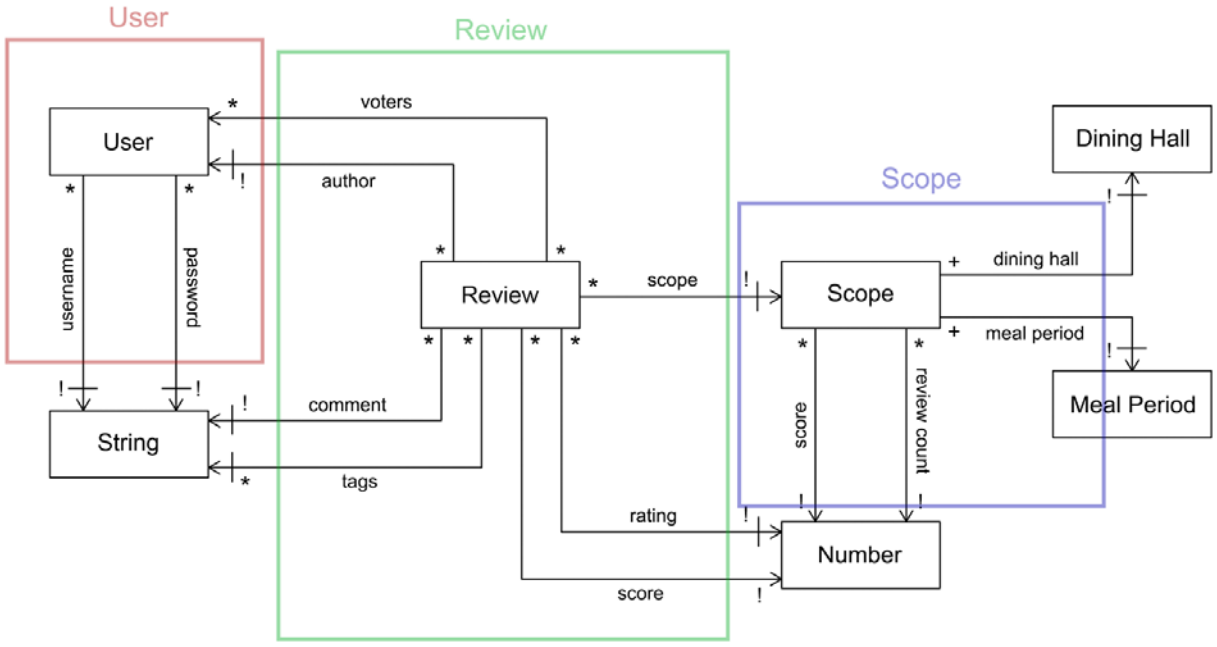
Votes (Upvote/Downvote): Users can vote on the helpfulness of reviews. This allows Foodback to display the most helpful reviews first. Naturally, votes are associated with specific reviews. They are closely related to the concept of **score**, which is the number of upvotes on a review minus the number of downvotes. Review scores allow for the highlighting of accurate portrayals of dining while weeding out inaccurate ones.

Tags: Users can optionally specify tags for their reviews when they wish to comment on something beyond the dining hall and meal period, such as when they want to discuss a particular dish. Tags are strings that are used to label a review; users can search for reviews with certain tags on them.

Data Model



Data Design



API Specification

POST /users/signup

Allows the user to sign up for a Foodback account. Upon success, the user is automatically logged in.

Request body:

- email: the email address to use for the account; must be @mit.edu
- password: the password to use for the account; must be <= 30 chars

Response:

- success: true if the user was created and logged in
- content: on success, an object with a single field 'username', which contains the username that has been granted to the new user (the part of their email address before the @mit.edu)
- err: on failure, an error message

POST /users/login

Logs the user into his or her Foodback account.

Request body:

- username: the user's username
- password: the user's password

Response:

- success: true if the user was logged in
- err: on failure, an error message

GET /users/logout

Logs the user out of his or her Foodback account.

Response:

- success: true if the user was logged out
- err: on failure, an error message

GET /reviews

Gets reviews from all dining halls and all meal periods, optionally filtered by tags.

Request query:

- (OPTIONAL) tags: a comma-separated list of tags to search for

Response:

- success: true if the user was created and logged in
- content: on success, an array containing the review objects that matched the search
- err: on failure, an error message

GET /reviews/:dininghall

Gets reviews from a specific dining hall, optionally filtered by tags.

Request parameters:

- dininghall: the name of the dining hall for which to find reviews

Request query:

- (OPTIONAL) tags: a comma-separated list of tags to search for

Response:

- success: true if the user was created and logged in
- content: on success, an array containing the review objects that matched the search
- err: on failure, an error message

GET /reviews/:dininghall/:mealperiod

Gets reviews from a specific dining hall and meal period, optionally filtered by tags.

Request parameters:

- dininghall: the name of the dining hall for which to find reviews
- mealperiod: the name of the meal period for which to find reviews

Request query:

- (OPTIONAL) tags: a comma-separated list of tags to search for

Response:

- success: true if the user was created and logged in
- content: on success, an array containing the review objects that matched the search
- err: on failure, an error message

POST /reviews/post

Allows a logged-in user to post a review.

Request body:

- hall: the dining hall that this review is for
- period: the meal period that this review is for
- rating: the star rating for the review, a number from 1-5
- content: the text content of the review
- (OPTIONAL) tags: a comma-separated list of tags to apply to the review

Response:

- success: true if the review was successfully submitted
- content: on success, an object with a single field 'review', which contains the review that was just posted
- err: on failure, an error message

GET /reviews/delete/:review_id

Allows the logged-in user to delete a review that he or she wrote.

Request parameters:

- review_id: a String representation of the MongoDB _id of the review

Response:

- success: true if the review was successfully deleted
- err: on failure, an error message

GET /reviews/vote/up/:review_id

Allows a logged-in user to upvote a review, provided he or she has not voted on the review before.

Request parameters:

- review_id: a String representation of the MongoDB _id of the review

Response:

- success: true if the vote was successfully submitted
- content: on success, an object with a single field 'score', which contains the new score of the review
- err: on failure, an error message

GET /reviews/vote/down/:review_id

Allows a logged-in user to downvote a review, provided he or she has not voted on the review before.

Request parameters:

- review_id: a String representation of the MongoDB _id of the review

Response:

- success: true if the vote was successfully submitted
- content: on success, an object with a single field 'score', which contains the new score of the review
- err: on failure, an error message

Design Challenges

*We will not choose solutions to some design challenges until we implement the client-side code. These design challenges are marked with ** (double asterisks).*

How do we authenticate users? We need a way to prevent spamming on the web application.For example, users outside of MIT should not be allowed to post comments/spam the real users with spam reviews. To this end, we require users to have an mit.edu email when they sign up as well as a password. Thus, to access sensitive areas of the application (particularly those that make changes to the database), we require that users be authenticated.

Chosen Solution:

- Allow users who have an MIT email to sign up for a Foodback account where they can submit/view feedback. They also have to specify a password to subsequently sign into the system. However, our implementation does not involve checking if the email address used is valid. Later on, we plan to add an email confirmation feature that ensures the user has access to the email address he specified. However, with this approach, users can also create mailing lists and use the mailing list address to create a Foodback account (which will generate "dummy" accounts). But at the same time, this approach enables users to create account names that allow them to not be identified.

Alternate Solution:

- Enabling users to use their MIT certificate to sign-in to the system. This allows users to sign-in without creating an account but we can also include the added feature of allowing users to also have a username. *Although we initially planned to use certificates for user authentication, we could not find a reasonable way to integrate MIT certificates with our Express app. After multiple unsuccessful attempts, certificate integration was found to be rather complicated when the website is on a non-MIT domain. We contacted scripts.mit.edu regarding Express / node.js support and received a response that node is installed but not supported. Two other user have managed to make it work but in a "hacky" way, not suitable for this project. We then tried using a self-signed certificate but none of our attempts was fruitful. Thus, we moved on to trying another authentication system instead.*

How can we guide the user to helpful reviews? A key idea is to keep it simple, so that users are encouraged to use Foodback. A user should be able to quickly make a decision based on the reviews, and for this to be possible, the reviews must be easy to go through.

Chosen Solution:

- Allow users to upvote and downvote reviews based on how accurate and useful they are. Upvoting a review will increase its score, and downvoting will decrease the score. Reviews can be displayed with highest scores first, so the user just needs to look at the top of the list to see the most helpful reviews.

Alternate Solution:

- Enable users to comment on reviews so that users can address particular things they liked or disliked about a review. A disadvantage of this approach is that the threads can become very long and cumbersome, defeating the purpose of enabling users to decide quickly. However, it allows users to see exactly what other diners agreed on or challenged.

Can reviews be edited or deleted? When a user leaves a comment card or submits to the online feedback form, that action cannot be taken back. However, it also (presumably) does not have permanent effects; comment cards are removed after a while, and feedback form submissions are never posted publicly in the first place. Foodback reviews should naturally be considered separately.

Chosen Solution:

- Allow deleting but not editing. This encourages users to post reviews freely, since they can delete the review if they change their mind. As a result, it may increase participation. Since a review-based site depends heavily on having a large group of participants, this is of utmost importance for our site.

Alternate Solution:

- Do not allow editing and deleting. This may simplify the implementation of the application. It also encourages users to think twice about what they write. But it might also deter any posting at all.
- Allow both editing and deleting: Enabling editing reviews would in some sense invalidate the upvotes/downvotes on that particular post. For example, a person might post a very positive review that gets a large number of likes, but later edit that review to make it negative, hence voters are not duped. This is especially important in the light that we will be filtering posts by usefulness based on the number of votes a review receives. It is essential that the content of a review and the rating be consistent in order to successfully implement this feature to simplify a user's decision.

**** Should information about a reviewer be publicly displayed?** Reviews will seem more personal and users will feel more accountable if we display some identifying information with reviews, such as a username or real name. However, users may not want to share this information.

Potential Solutions:

- Do not display any identifying information. This is the most effective for protecting user information. Users will feel more comfortable leaving honest reviews if they know that any offended parties cannot identify them.
- Display the user's first name. We would have to collect this information from users upon signup. However, it provides a reasonable amount of accountability without exposing too much personal information.
- Display the user's username. This maximizes accountability because each user's username is his or her Kerberos, and therefore readers can find out who they are using the MIT people directory.

How many reviews can a user submit? If a particular user submits many reviews, it is possible for them to dominate the review system and make their opinion appear to be more prevalent than it really is.

Chosen Solution:

- Do not limit reviews. If a user submits many reviews, they are probably very passionate about the topic of MIT dining. We may actually want them to have more weight in the discussion because they might be better informed than other users. This also prevents us from having to add concepts of time to our data model.

Alternate Solutions:

- Limit reviews to a certain number per day. This is a relatively simple method of limiting reviews, but would still require us to include some form of comparable timestamping in the review model to facilitate checking the user's quota.
- Limit reviews to one per user per meal period each day. A user may very well have something to say about the dining halls during each meal period. We should allow the posting of that number of reviews. By allowing no more than that many reviews, we can reduce spam and allow other users' voices to be heard. This would be the most complex to implement.

How should reviews be categorized? For reviews to be useful to other users, there must be a way for users to find them in a structured, non-convoluted way.

Chosen Solution:

- Require a scope for each review, and allow extra user-specified tags. The scope, which consists of a dining hall and meal period, ensures that reviews are restricted to the time and place for which they are relevant. Tags allow users to comment on more specific things like dishes.

Alternate Solutions:

- Put each review in a category. This is a simple method that could greatly simplify searching for reviews. However, we would need a large number of categories to accommodate all types of reviews that users might submit, and these categories would not have roughly the same number of reviews; users might comment on the dining hall as a whole, or on a specific dish.
- Put each review in multiple categories. This provides greater flexibility, but has a major drawback: most dining halls operate independently. They have different chefs, equipment, etc. Thus, a negative review on food without specifying which dining hall the meal was had in would negatively affect other dining halls, an unwanted side effect.

How do we keep track of votes? We will be using votes to give reviews scores so that we can sort them and display the best ones first. We need to keep track of votes in such a way that users cannot vote more than once on a given review.

Chosen Solution:

- Do not include votes in the database; instead, simply use scores. Reviews have a score field that represents their current numeric score. When a vote is received, the field is updated to reflect

the new score. To prevent votes from multiple users, reviews also maintain a list of users who have voted on them, and this can be cross-checked with the user who is attempting to vote in order to validate the action. This has a side effect that votes now become immutable and the user cannot decide to take his vote back.

Alternate Solutions:

- Have the user objects contain lists of vote objects. The vote objects themselves would refer to the reviews on which the votes were cast. This would make it easy for us to look up which reviews a user has cast votes on, and display these on a page if the user so chooses. However, this adds complexity and is not a core feature for the app.
- Have the review objects contain lists of vote objects. The vote objects would refer to the users who cast the votes. This would make it easy for us to look up which users have voted on a review, and to provide a detailed breakdown of up/down votes on a given review. But like the previous suggestion, this adds complexity without facilitating a core feature.

Sources

1. Bootstrap
2. Project 3 demo utils