A3 Design Notebook

Concept Design Models

We first introduce each concept, followed by its purpose, actions, operational principles and state. The state of all the concepts will also be combined into a single diagram for more comprehensive details. The constraints of the data model will be introduced after the diagram.

Concept: following

Purpose: For a user to label other users so that he/she can easily find them and their posts (which include Freets and Refreets).

Actions:

addFollowing (u, u’: User):

u.followings = u.followings + u’

removeFollowing (u, u’: User):

u.followings = u.followings - u’

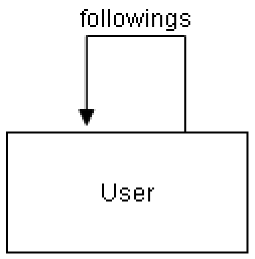
Operational principle:

Before performing addFollowing (u, u’), u’ would not be in u.followings. Hence, at the same time, if a User u has never performed addFollowing (u, u’) for any u’, u.followings would be empty.

After performing addFollowing (u, u’), u’ would be in u.followings. Further, u would be able to find u’ by querying u.followings, and then view the posts of u’ easily.

After performing removeFollowing (u, u’), u’ would no longer be in u.followings. Note that if u’ were not in u.followings before performing removeFollowing (u, u’), the action would do nothing.

State:



Concept: upvoting

Purpose: For a user to label posts (which include Freets and Refreets) so that he/she can easily find the labeled posts.

Actions:

upvote (u: User, f: Freet):

u.upvotedFreets = u.upvotedFreets + f

revokeUpvote (u: User, f: Freet):

u.upvotedFreets = u.upvotedFreets - f

Operational principle:

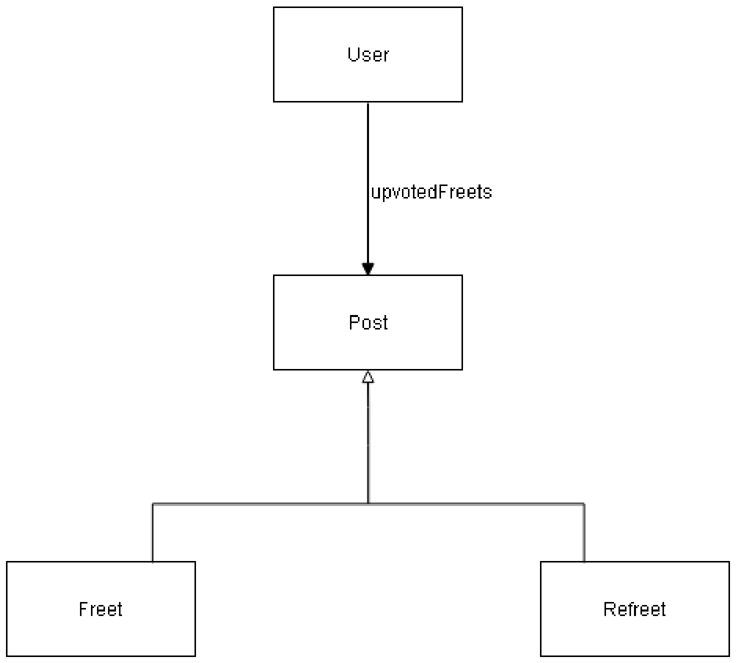
Before performing upvote (u, f), f would not be in u.upvotedFreets. Hence, at the same time, if a User u has never performed upvote (u, f) for any f, u.upvotedFreets would be empty.

After performing upvote (u, f), f would be in u.upvotedFreets. Further, u would be able to find f by querying u.upvotedFreets, and then view f easily. Since u.upvotedFreets is a set, if f is already in u.upvotedFreets, performing upvote (u, f) again would do nothing.

After performing revokeUpvote (u, f), f would no longer be in u.upvotedFreets. Note that if f were not in u.upvotedFreets before performing revokeUpvote (u, f), the action would do nothing.

All of the above actions and operational principles can be applied to Refreets by substituting the Freet f into a Refreet r. We only write them once out of simplicity consideration.

State:



Concept: refreeting

Purpose: For a user to post and optionally comment on other users' Freets, while showing the original author and content of the shared Freets.

Actions:

refreet (u: User, f: Freet):

New r: Refreet;

r.originFreet = f;

r.originAuthor = f.author;

r.author = u;

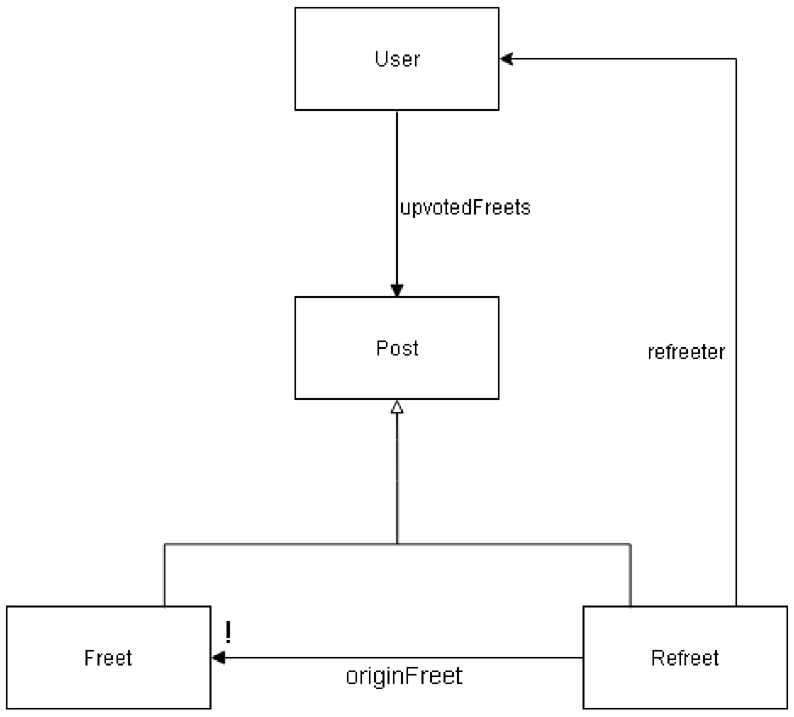
delete (r: Refreet):

Remove Refreet r from the set of Refreets.

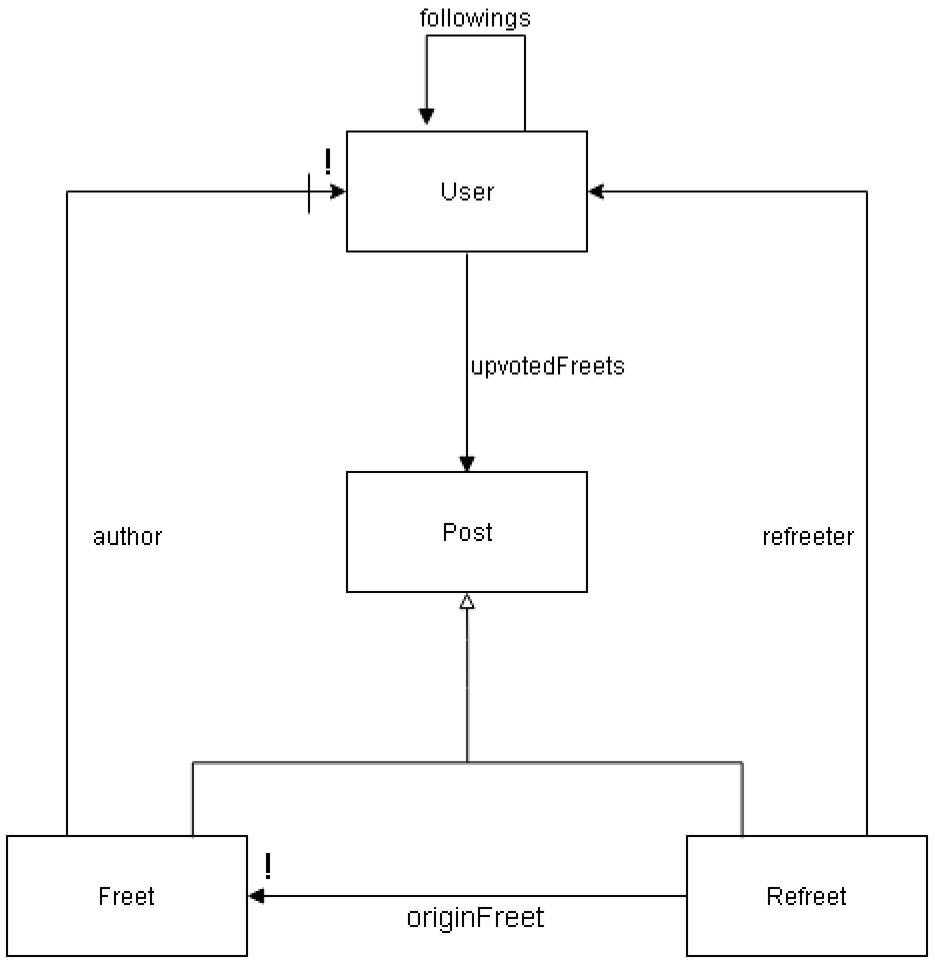
Operational principle:

The user performing refreet (u, f), namely u, will be assigned as the r.author since u creates r by performing refreet (u, f). In this way, r will be shown as a post of u if findByAuthor(u) is performed. Note that r.author differs from the author of the originFreet of r, namely f.author, and the author of f will be stored in r as r.originAuthor.

State:



Comprehensive Data Model:



Constraints:

no iden[User] & followings (users cannot follow themselves)

(Data model diagrams are generated using: https://app.diagrams.net/**)**

Design Reflections

1. Many-to-Many Following Relationship

We treated the Users' relationship with relation followers and followings. The two relationships are many-to-many. One is allowed to follow multiple users and have multiple followers.   A user can be in both sets, i.e. users can follow each other. By using Users.followers, we can get a list of followers of the users. By using Users.followings, we can get a list of users the current user is following. Unfollowing someone will automatically remove him/her from the current user's following list, and remove the current user from his/her follower list. In our Profile page, users are allowed to view and manage the following relationships. Users can follow and unfollow other users when looking through Freets. Users are not allowed to follow themselves so we designed the follow button after the current username would disappear when the user is signed in. One limitation is that following users through a Refreet is not supported and users cannot click through the following relationship to view the profile of the following users.

Alternatives Considered: Initially, we only considered to have a unilateral relationship between users, that is we only have Users.followings. But we think this single relation is not enough. Another alternative is to allow users to follow other users through Refreets. This would be a slight revision to our current implementation of Refreets. This could potentially be desirable since a user might be interested in following other users who share the same topics or Freets that he or she refreeted or saw others refreeted. We would consider implementing this in the future.

2. Defining Refreet as a Subclass of Freet
In our design, we made our decision that we would let Refreet be part of the Freet concept. We design a subtype Refreet by simply adding more properties to Freet and sharing some methods. We called all posted Refreets and Freets as Posts. To avoid duplicate storage, in our design, Refreet doesn’t store all the content of the original post, but only the id of it. We then use an API to get the content of the original post. By doing so, not only do we avoid the need for additional storage space, but we also ensure that Refreets are synchronized with the original post, meaning that when the author modifies the original post, Refreets will change as well. Users are allowed to comment on the original Freet when refreeting. If the original Freet is deleted, we will see a message on the Refreet side saying that the original Freet was deleted, but the Refreet remains, and the Refreeter can choose whether to delete it or not. We also allow users to refreet their own posts as they may want to refer to an old post at some point. One limitation in our design is that in the Freet class, we actually have both Freet and Refreet. This makes it hard to refreet a refreet as the refreet() method is specified for Freet. We are in progress to merge Refreet and Freet together, i.e., we would use a single method to create both Freet and Refreet. Another limitation is that as we allow users to refreet as many times as they want,as we assume they may leave different comments for each Refreet, this may lead to redundant similar refreet posts.

Alternatives Considered: Initially, we made two seperate classes, one for Freet and one for Refreet. As we intended to make a new class Refreet, we realized there are too many similar implementations between Freet and Refreet. It does not make sense to break some functionality into two separate concepts because they share similar purposes, actions, state etc.

3. UpvotedFreets containing both Freets and Refreets

upvotedFreets is the relation between users and posts. We defined it as a property of Users type. It is a set containing all the Freets and Refreets the user upvoted. We also added the corresponding properties for Freets class, the upvoters as a list and the numUpvotes to count. Users are allowed to upvote each post once, that means they cannot appear multiple times in a Freet’s upvoters list. Once the current user upvotes a post, the upvote button will change to ‘unupvote’, which allows users to revoke an upvote using the same button. Since we believe some refreeters would give interesting comments that some other users may agree with, we also designed that users can also upvote a Refreet, and the upvotes counts for the Refreet, stored in the same way as a Freet. Users are allowed to view all the upvoted Freets through their profile.

Alternatives Considered: The other alternative would be to disallow users to unupvote Freet after they upvoted and allow users to upvote a Freet as many times as they want. But we think it is unfair to not allow a user to revoke an upvote as they might misclick ‘upvote’ or change their mind later. An alternative for allowing a user to upvote a post once is to allow multiple upvotes. This alternative does not match what most social media design for their user posts. Also, since we implemented upvotedFreets for each user as a set, performing upvote multiple times seems meaningless.

4. User interface design
Display Freets as collections in Profile page: Upon adding the concept of Refreeting, Following and Upvoting, we need to have somewhere to display this extra information related to a User. In the userProfile page, users are allowed to make modifications like editing and deleting to their posts, followings and Likes(upvoted Post) without reaching out to the post lists on the Freet page. They are also allowed to manage their followings and see their followers. We also intentionally calculated the total number of each collection and displayed it on the buttons. One limitation is that, in our design, the user profile page is private and visible only to the currently logged-in user.

List Freets and Refreets separately: On the main page where users can view all the posts, we listed the Freets and Refreets in separate columns. This could help the users understand what contents are original and what are other people’s comments on the original contents. This is also easier to implement since we implement Refreets as a subclass of Freets.

Alternatives Considered:
Display Freets as collections in Profile page: Alternatively, we considered adding more modules on the main Freet page to display user-related information. One big problem is that the Freet page looks too messy and not easy to understand. So we decided to have a separate page to display personal information. We also considered calculating all the upvotes that received by the same author, and display it on the user page. As mentioned before the exclusivity of the user profile, we will let the user decide if they want to come public with it, by adding more features to the settings page. And we considered making all followings and followers' names clickable, then users will be able to click through names to see other user’s profiles.

List Freets and Refreets separately: an alternative would be to list them together. It is also possible to have another column listing all the posts whose authors are followed by the current user. This would introduce duplicate content on the main page and might be redundant and the column containing all the posts might be too long. A potential revision is to sort posts by the number of upvotes before displaying. This can save users some time if they are just interested in the most popular ones. We shall consider this option in the future.

Social/Ethical Reflections

1. Allowing users to revoke their actions
Users are allowed to revoke an upvote and unfollow a user in case they accidentally conduct the actions or they change their mind. Since users would use upvoting to store the posts and use following to store users that they want to view at a later time. It would be reasonable to allow for them to keep their listing dynamic and flexible. Another case could be that the upvoter no longer agree with the original Freet if the original Freet is modified by the author or the upvoter has a changed opinion. On the other hand, since the number of upvotes is displayed for each post as an indication of its popularity, it would be helpful to reflect the change of popularity over time by allowing unupvote if there is a change of content or change of people’s opinion.

2. Authentication and Ownership
All the user-related behaviors like refreeting, following, and upvoting require anthentication as we don’t want our users being irresponsible for their behaviors when using Fritter. In our design, for each Freet, we also display the users who upvoted or refreeted the post. This may not guarantee the privacy of users when they do not want to let others know that they upvoted or refreeted a post. This may be mitigated by allowing users to be anonymous publicly but administrators will still keep tracking their posts at the backend. We also respect the ownership of the authors, i.e. the creator of the contents. Only the author can modify the Freet. When they decide to permanently delete their post, all the Refreets derived from it would not see the original Freet as well but only see a message saying the original content has been deleted. This also helps the administrators of the site to delete inappropriate Freet: the administrators can just delete the original post without having to go looking for the Refreets.

3. Allowing Freets modification may misrepresent the Refreets
Users are allowed to make modifications to their past posts. This gives Freet authors some flexibility in case they made a mistake in their Freet or they said something inappropriate and would like to revise it. However, this could cause some confusion in the Refreets. For instance, if an original Freet said something wrong and a number of Refreets pointed it out, but then, the author of the original Freet edited the wrong content, late viewers of the Refreets might be confused and since they could no longer find the wrong content in the original Freet. To mitigate this confusion, we could display a notice saying “the original content was modified” after the author revised his or her original Freet.