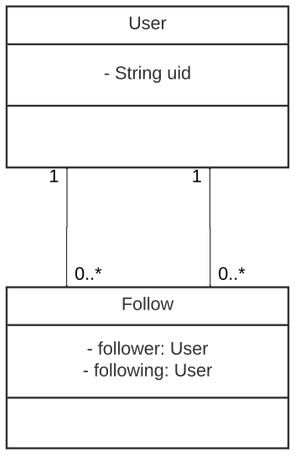
Implementation Analysis

Zhaoyi Zhuang | February 20, 2022

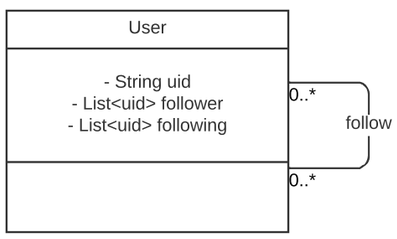
Option A
Create a follow lookup table to represent the relationship between users.

It the most versatile, It can be implemented in both relational and non relational database. It is also doable in OOP design. It might have complicate joining in a non relational database, but this complexity is less hornorous than maintaing synchronous arrays in option B. Also option B would require a transaction to make two updates atomically.



Option B
Each user caontains a list of reference to users who follow him/her, and a list of reference to users who are followed by him/her.

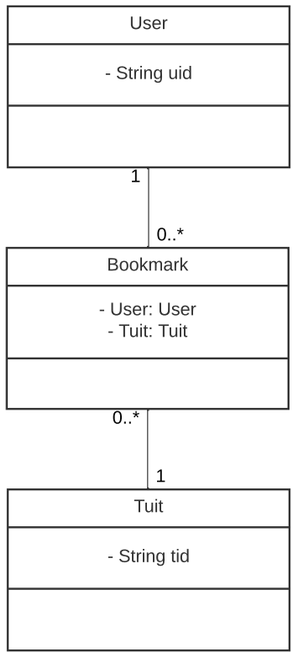
This solution cannot be implemented in relational database becuase it requires multi-valued column. But it is doable in OOP design. But, it could be a difficulty to maintain the consistent on both sides.



Based on my understanding, Option A will be more efficient when users follow and unfollow others more often, while option B is more efficient when users check their following and follower list more often.
In addition to the analysis I made in Option A. I will go with Option A for Follows API

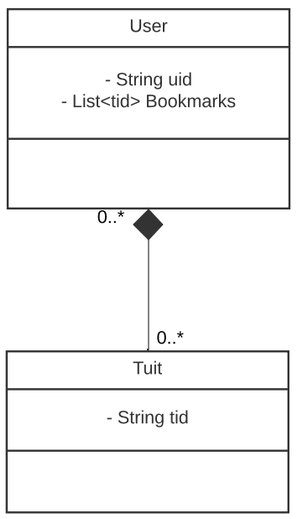
Option A
Create a Bookmark lookup table to represent the relationship between user and tuit.

It the most versatile, It can be implemented in both relational and non relational database. It is also doable in OOP design. but might complicate joining in a non relational database.



Option B
Each user caontains a list of reference to tuits that is bookmarked by the user.

This option is undoable in relational database.
However, since we are using MongoDB and oop design, this is a feasible implementation.
Also, bookmarks do not have any other relations with any other objects. So, this is a good choice.
The challenge here will be adding and deleting the tuits into and from the bookmarks



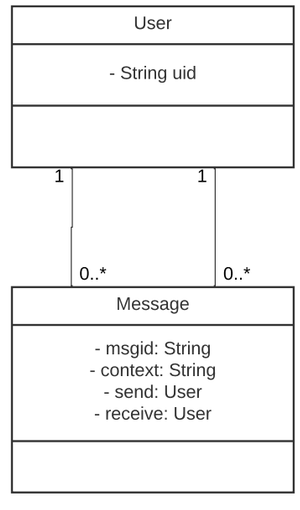
Option A will be efficient when add and delete bookmarks, but relatively slow when retrieve a list of tuits.
Option B should be good at all operations. But there is a challenge of delete and add tuit to collection.

At this time, I will go with Option B, because it is better solution to me.

Option A
Create a Message lookup table to represent the relationship between users. Each Message object is one single message.

It can be implemented in both relational and non relational database. It is also doable in OOP design. It might have complicate joining in a non relational database.

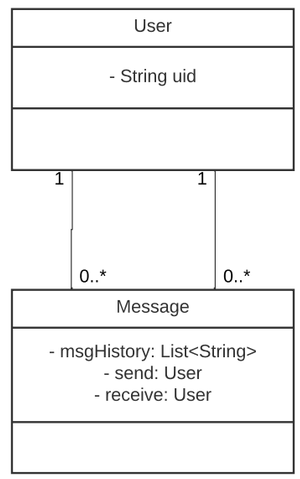
It is easy to delete and view a specific message. Lke how email works



Option B
Create a Message lookup table to represent the relationship between users. However, at this time, Each Message object contain all message that a user sends to another user.

It might caused unnormalize in relational database due to multi-value in msgHistory. But It is doable in our cases. It might have complicate joining in a non relational database. Also, It might be a challenge to locate a specfic message.

It is more efficient when view a list of message users send and receive.



Option A is a more straightfoward implementation to me. It is very versatile and it works more closely to homework's requirements where messages act more like emails.
Option B might be a good choice if user want to build converstation with another user and want to see the history of their converstaion easily.

Thus, for this assignment, I will go with Option A.