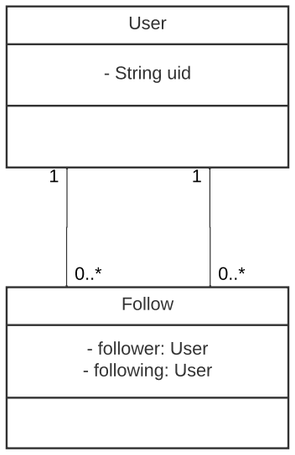
Implementation Analysis

Zhaoyi Zhuang | February 20, 2022

Follow

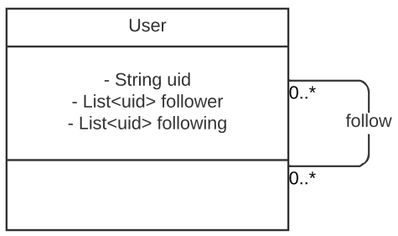
Option A
Create a follow lookup table to represent the relationship between users.

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.

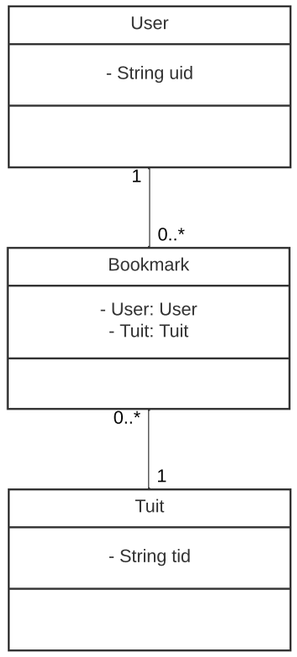


Based on my understanding, Option A will be more efficient when users follow and unfollow others more often, while option B is more efficient when users check their following and follower list more often.
In addition to the analysis I made in Option A. I will go with Option A for Follows API

Bookmark

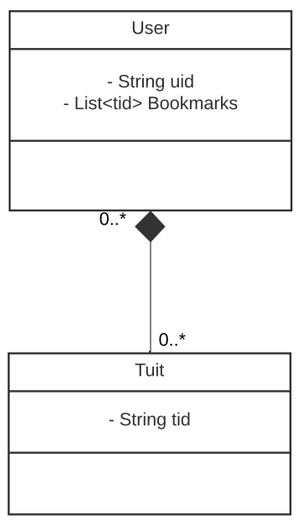
Option A
Create a Bookmark lookup table to represent the relationship between user and tuit.

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.

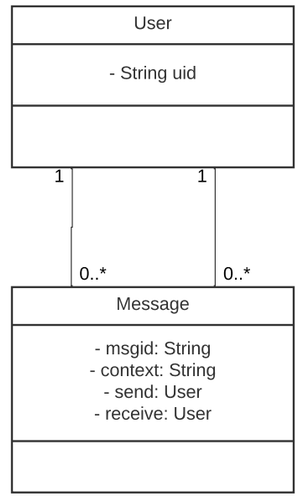
At this time, I will go with Option B, because it is better solution to me.

Message

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.

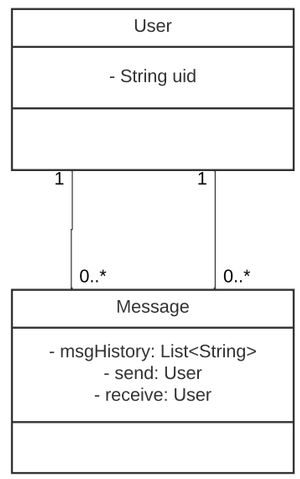
It is easy to delete and view a specific message. Lke how email works



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.

Follows API:

Name: User follows another user
Analysis: If User Bob follows User Jack, Jack will be added to Bob's following list, and Bob will be added to Jack's follower's list.
HTTP method: POST
Path pattern: /users/:uid/follows/:followingid
Request: Follower User's PK, Following User's PK
Response: New Follows JSON

Name: User unfollows another user
Analysis: If User Bob unfollows User Jack, Jack will be removed from Bob's following list, and Bob will be removed from Jack's follower's list.
HTTP method: DELETE
Path pattern: /users/:uid/follows/:followingid
Request: Follower User's PK, Following User's PK
Response: Delete status

Name: User views a list of other users they are following
Analysis: User Bob will see a list of other users that Bob followed before.
HTTP method: GET
Path pattern: /users/:uid/follows
Request: User's PK
Response: User JSON Array

Name: User views a list of other users that are following them
Analysis: User Bob will see a list of other users who are fans of Bob.
HTTP method: GET
Path pattern: /users/:uid/follows/fans
Request: User's PK
Response: User JSON Array

Name: User removes one follower from their following lists
Analysis: If User Bob do not like User Jack maybe due to some privacy reason, Bob can remove Jack from his fans list so that Jack is no longer following Bob.
HTTP method: DELETE
Path pattern: /users/:uid/follows/fans/:followingid
Request: Follower User's PK, Following User's PK
Response: Delete status

Name: User follows another user secretly
Analysis: If User Bob wants to follow User Jack. But Bob does not want Jack to know this, then Bob can follow Jack secretly. As the result of it, only Bob can see Jack in his following list. User Peter cannot see Jack in Bob's following list. However, Jack's follower number will

count Bob. For example, if Jack does not have any other follower expect Bob, Jack's follower number will be 1 even Jack would not know who is this.
HTTP method: POST
Path pattern: /users/:uid/follows?tagid=-2/:followingid
Request: Query parameter tagid = -2, Follower User's PK, Following User's PK
Response: New Follows JSON

Bookmarks API:

Name: User bookmarks a tuit
Analysis: When a user likes a Tuit and wants to make a special mark of that tuit, the user can bookmark that tuit. In the future, users can view the tuits they bookmarked earlier easily. The HTTP method is PUT because users are putting tuits into bookmark list instead of creating a bookmark relationship instance. So, bookmarking a tuit is actually update user's bookmark list.
HTTP method: PUT
Path pattern: /users/:uid/bookmarks/:tid
Request: User's PK, Tuit's PK
Response: Update status

Name: User unbookmarks a tuit
Analysis: If a user is no longer interested in a tuit or the user would not need it anymore in the future, the user can unbookmarks the tuit. This tuit will be removed from the user's bookmark list. The HTTP method is PUT because of the same reason as above.
HTTP method: PUT
Path pattern: /users/:uid/bookmarks/remove/:tid
Request: User's PK, Tuit's PK
Response: Update status

Name: User views a list of tuits they have bookmarked
Analysis: If a user wants to see tuits they bookmarked, they can view a array of tuits.
HTTP method: GET
Path pattern: /users/:uid/bookmarks
Request: User's PK
Response: Tuit JSON Array

Name: User empty their bookmark lists
Analysis: If the user thinks there are just too many bookmarks, and the user does not want to delete all bookmarks one by one manually. Then there is an option for them to empty their bookmarks list by click one button easily.
HTTP method: DELETE
Path pattern: /users/:uid/bookmarks
Request: User's PK
Response: Delete status

Name: User view a single bookmark
Analysis: If user wants to view a specific bookmark, he/she can choose one tuit among all tuits in his/her bookmark list. Then the user can view the tuit he/she interested in.
HTTP method: GET
Path pattern: /users/:uid/boomakrs/:tid
Request: User's PK, Tuit's PK
Response: Tuit JSON

Messages API:

Name: User send a message to another user
Analysis: If a user wants to send a private message to another user without been seen by other users, the user could use direct message function. The message works like a email that only send and the receiver can see this message.
HTTP method: POST
Path pattern: /users/:uid/messages/:receiveid
Request: User's PK, message in body
Response: New Message JSON including PK

Name: User views a list of messages they have sent
Analysis: A user can see all messages he/she sent to other users.
HTTP method: GET
Path pattern: /users/:uid/messages/sent
Request: User's PK
Response: Messages JSON Array

Name: User views a list of messages sent to them
Analysis: A user can see all messages he/she received from other users.
HTTP method: GET
Path pattern: /users/:uid/messages
Request: User's PK
Response: Messages JSON Array

Name: User deletes a message
Analysis: If a user feels that there are too many messages, or the user wants to delete some message for some reasons. The user can delete it, and the message will be removed from both side. Neither the sender and the receiver can see this message any more.
HTTP method: DELETE
Path pattern: /messages/:msgid
Request: Message's PK
Response: Delete status

Name: User views a message

Analysis: A user can view a specific message from the list of messages the user received and sent.
HTTP method: GET
Path pattern: /messages/:msgid
Request: Message's PK
Response: Message as JSON

Name: User pins a message
Analysis: User can pin a message that the message will be top of user's view list.
HTTP method: PUT
Path pattern: /messages?pin=1/:msgid
Request: Query Parameter pin = 1, Message's PK
Response: Update status