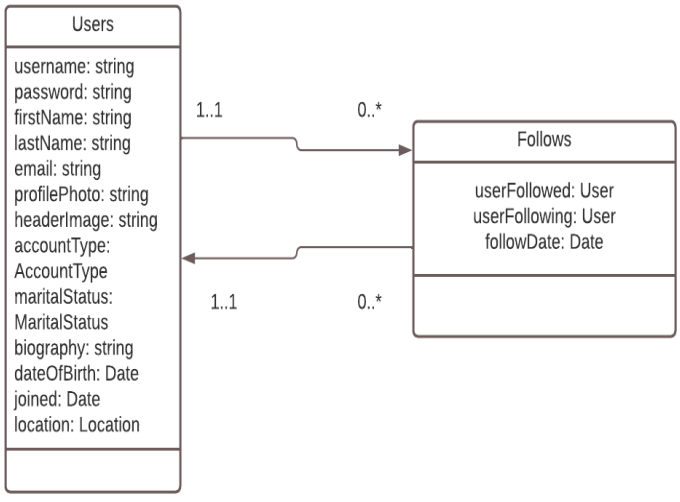
Assignment A2

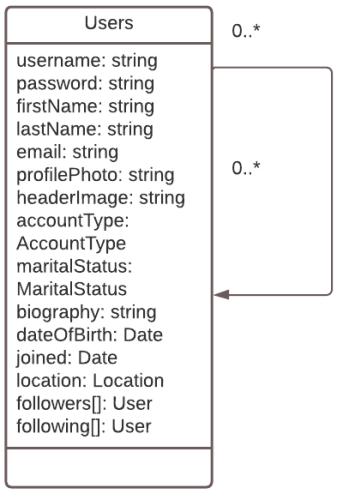
Follows:

Use Case	*Method /resource/path*	*Request*	*Response*
1. ***A user Follows another user***	POST /api/users/:uid1/follows/:uid2	User's and other user's uid PK	New Follow JSON including PK
2. ***A user Unfollows another user.*** 3. ***A user Unfollows all the user.***	DELETE /api/users/:uid1/follows/:uid2 /api/users/:uid1/follows/	User's and other user's uid PK	JSON array containing delete count.
4. ***User Views list of other users they are following.***	GET /api/users/:uid/follows	User's uid PK	Follow JSON array with users populated.
5. ***User Views list of other users that are following them.***	GET /api/users/:uid/followers		Follow JSON array with users populated.
6. ***User Views list of recent users that are following them.***	GET /api/users/:uid/recentFollowers		Follow JSON array with users populated.

Current Design:



Design Alternative:



Current Design: In the current design the follows relationship is a different collection, which makes the CRUD operation quite simple.

Furthermore, it gives us the advantage of manage Follow relationship, such as adding extra attribute without affecting any other entity.

Alternative design: In alternative design the followers and following users stay at the same table.

The crud operation becomes extremely difficult as one has to manage two arrays to commit even a minor change.

Description: Use case

1. ***A user Follows another user***
   In the above use case, a user can click on the follow button of another user.
   POST /api/users/:uid1/follows/:uid2
   Here uid1 is users id
   Uid2 is uid of the person user 1 is following.
   The above request will create a record in follows collection in the data base which can be uniquely identified by uid1 and uid2.
   The response consists of a follows JSON with user1 and user2 ids.
   Using the above post request user can follow the other user.

2. ***A user Unfollows another user***
   In the above use case, a user can click on the unfollow button of another user.
   DELETE /api/users/:uid1/follows/:uid2
   Here uid1 is users id
   Uid2 is uid of the person user 1 is following.
   The

   above request will delete a record in follows collection in the data base which can be uniquely identified by uid1 and uid2.
   The response consists of a follows JSON with delete count.
   Using the above delete request user can unfollow the other user.

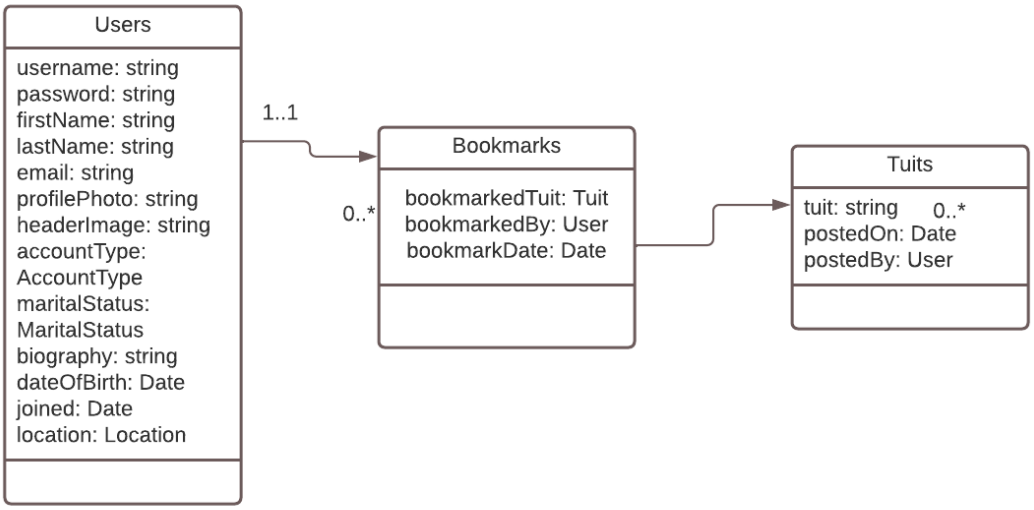
3. ***User Views list of other users they are following.***
   In the above use case, a user can click on the following button to see the list of users they follow.
   GET /api/users/:uid/follows
   Here uid is users id.
   The above request will fetch records from follows collection in the data base which can be uniquely identified by uid1.
   The response consists of a follows JSON array with User details populated.

4. ***User Views list of other users that are following them.***
   In the above use case, a user can click on the followers button to see the list of followers.
   GET /api/users/:uid/followers
   Here uid is users id.
   The above request will fetch records from follows collection in the data base which can be uniquely identified by uid1.

The response consists of a follows JSON array with User details populated.

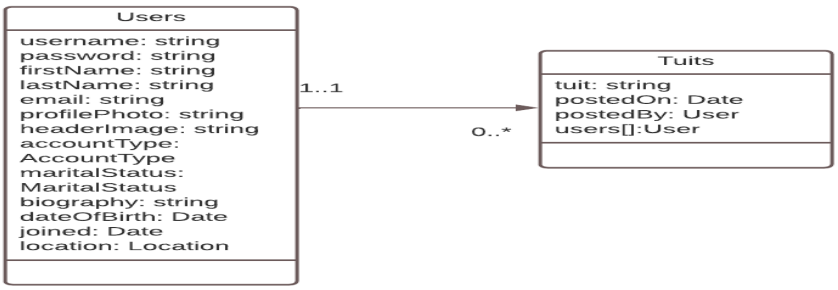
5. ***User Views list of recent users that are following them..***
   In the above use case, a user can click on the recentFollowers button to see the list of followers who followed the user in last 2 hours.
   GET /api/users/:uid/recentFollowers
   Here uid is users id.
   The above request will fetch records from follows collection in the data base which can be uniquely identified by uid1.
   The response consists of a follows JSON array with User details populated. Only those users will be populated who followed the user in last 2 hours.

Bookmarks:



Current Design: In the current design the Bookmarks relationship is a different collection, which makes the CRUD operation quite simple.

Furthermore, it gives us the advantage of managing Bookmarks relationship, such as adding extra attribute without affecting any other entity.



Alternative design: In alternative design the users who bookmarked a tuit stays at the many side of the relationship. Tuits maintains ab array of users to identify bookmark operation.

The crud operation becomes extremely difficult as one has to manage an arrays to commit even a minor change.

Use Case	*Method /resource/path*	*Request*	*Response*
1. User Bookmarks a tuit,	POST /api/users/:uid/bookmarks/:tid	User uid and tuit tid PK	New BookmarkJSON including PK
2. User unbookmarks a tuit**.**	DELETE /api/users/:uid/bookmarks/:tid	User uid and tuit tid PK	JSON array containing delete count.
3. User views a list of tuits they have bookmarked	GET /api/users/:uid/bookmarks	User's uid PK	Bookmark JSON array.
4. User views a list of tuits they have recently bookmarked **.**	GET /api/users/:uid/recentBookmarks		Bookmark JSON array.

Description: Use case

1. ***A user Bookmarks a Tuit***
   In the above use case, a user can click on the bookmark button on the tuit.
   POST /api/users/:uid/bookmarks/:tid
   Here uid is users id and tid is tuit id which use wants to bookmark.
   The above request will create a record in Bookmark collection in the data base which can be uniquely identified by uid and tid.
   The response consists of a Bookmark JSON with user and tuit ids.
   Using the above post request user can bookmark a tuit.

2. User unbookmarks a tuit**.**
   In the above use case, a user can click on the unbookmark button on the tuit.
   DELETE /api/users/:uid/bookmarks/:tid
   Here uid is users id
   Tid is the tuit ID.
   The above request will delete a record in Bookmark collection in the data base which can be uniquely identified by uid and tid.

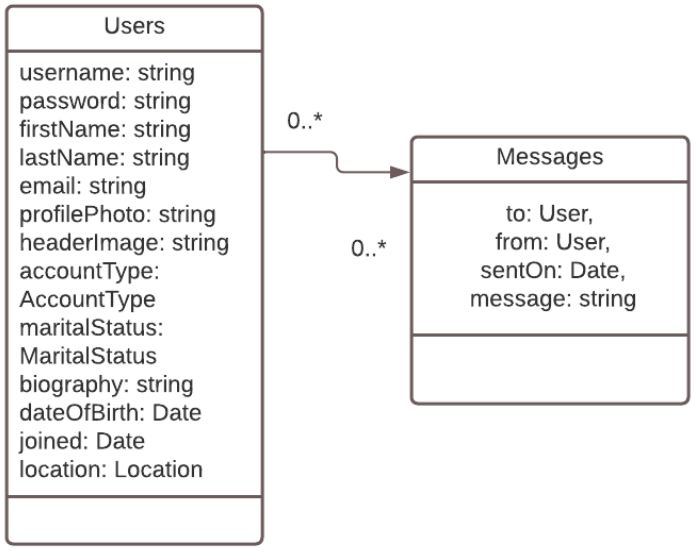
The response consists of a Bookmarks JSON with delete count.
Using the above delete request user can un bookmark a tuit.

3. User views a list of tuits they have bookmarked**.**
   In the above use case, a user can click on the bookmark button in his profile to see the list of users he bookmarked.
   GET /api/users/:uid/bookmarks
   Here uid is users id.
   The above request will fetch records from Bookmarks collection in the data base which can be uniquely identified by uid.
   The response consists of a Bookmark JSON array with User details populated.

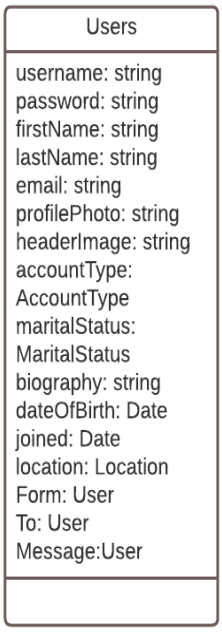
4. User views a list of tuits they have recently bookmarked **.**
   In the above use case, a user can click on the recentBookmark button in his profile to see the list of followers.
   GET /api/users/:uid/recentBookmarks
   Here uid is users id.
   The above request will fetch records from Bookmark collection in the data base which can be uniquely identified by uid.
   The response consists of a follows JSON array with User details populated. Only 2 hrs recent Bookmarks will be populated.

Messages

Current Design



Alternate Design



Current Design: In the current design the Messages relationship is a different collection, which makes the CRUD operation quite simple.

Furthermore, it gives us the advantage of managing Messages without need to have any other entities.

Alternate Design:

In alternate design we have created three attributes in user table, to, from and message. The design is not good as it created a lot of records and makes CRUD operations extremely difficult.

Use Case	*Method /resource/path*	*Request*	*Response*
1. User sends a message to another user	POST / api/users/:uid1/messages/:uid2	Message JSON ,User1 uid and user2 uid PK	New Message JSON including message PK
2. User deletes a message	DELETE /api/users/:uid1/messages/:uid2	User1 uid and user2 uid PK	JSON array containing delete count.
3. User views a list of recent messages sent to them	GET /api/users/:uid/recentMessages	User's uid PK	Message JSON array.
4. User views a list of messages they have sent	GET /api/users/:uid/Messages		Message JSON array.
5. User views a list of messages they have received	GET /api /users/Messages/:uid		Message JSON array.
6. User views a list of messages they have received by a particular user.	GET /api/users/:uid1/messages/:uid2	User1 uid and user2 uid PK	Message JSON array.

Description: Use case

1. User sends a message to another user
   In the above use case, a user can write a message and click on the send button to send a message.
   POST / api/users/:uid1/messages/:uid2
   Here uid1 is senders id and uid2 is receivers idto whom the message is being sent.
   The above request will create a record in Message collection in the data base which can be uniquely identified by uid1 and uid2.

The response consists of a Message JSON with user ids and message JSON.

2. User Delets a message.
In the above use case, a user can click on the delete message button on the delete button of message.
DELETE /api/users/:uid1/messages/:uid2
Here uid is users id 1
The above request will delete a record in Message collection in the data base which can be uniquely identified by uid1 and uid2.
The response consists of a Message JSON with delete count.
Using the above delete request user can detele a message send by another user.

3. User views a list of Messages they have Received **.**
In the above use case, a user can click on the messages button in his profile to see the list of messages.

GET /api /users/Messages/:uid

Here uid is users id.
The above request will fetch records from messages collection in the data base
The response consists of a Message JSON array with message details populated.

4. User views a list of messages they have sent
In the above use case, a user can click on the messages button in his profile to see the list of messages he received.
GET /api/users/:uid/Messages
Here uid is users id.
The above request will fetch records from messages collection in the data base
The response consists of a Message JSON array with message details populated.