Designing the Follows, Bookmarks, and Messages Restful Web Service APIs (30pts)

Design RESTful Web services for the ***follows***, ***bookmarks***, and ***messages*** resources. In addition to the use cases we provide, come up with two additional use cases and design a RESTful Web service API for each of the use cases.

Designing the Many to Many Follows RESTful Web Service API (10pts)

Following the principles demonstrated in the earlier examples, design a RESTful Web service for the ***follows*** resource for the use cases shown below and based on the class diagram provided here on the right.

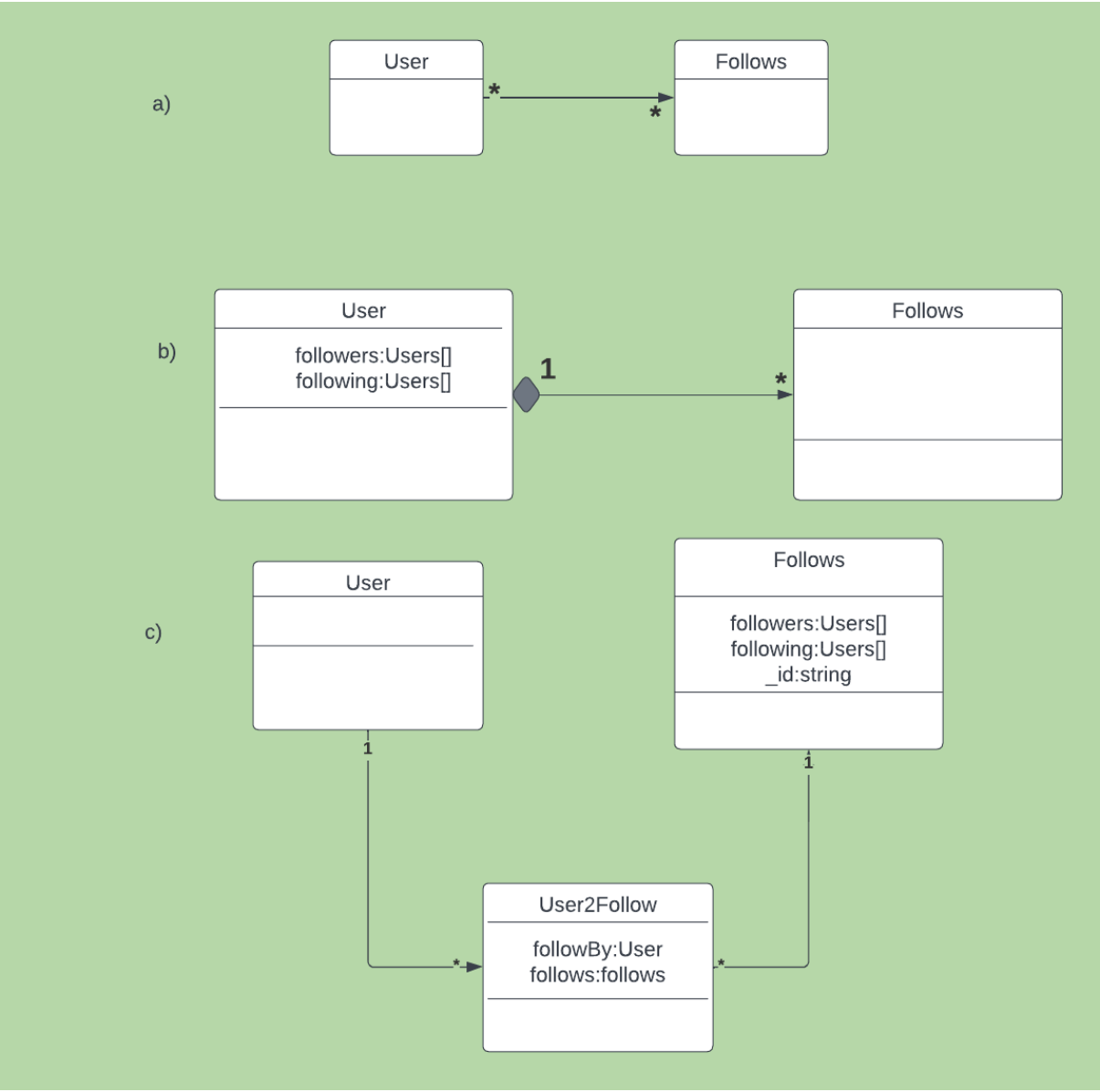
- User follows another user
- User unfollows another user
- User views a list of other users they are following
- User views a list of other users that are following them
- User retrieves a following user
- User retrieves a follower user

Nouns	***Verbs***	Verb phrases
a ***follow*** many ***follows***	***to follow***, follows followed, following	user follows a user, user is following a user user is followed by a user, user unfollows a user

Identifying CRUD operations

Create	***Retrieve/Read***	***Update***	***Delete***
users follow users	***users view followers***		***users unfollow users***

Implementation alternatives and analysis



Option (a) doesn't help. It just declares that there's a one to many relationship between users and follows. We know that already.
Option (b) suggests that we could store the followers and following users in arrays. This option is not clear on whether we would store an actual object in the array or merely references to the user's objects stored in some other collection. If we'd be using a relational database, this option would be impossible to implement. If this is an object oriented implementation then the array would clearly contain pointers to instances. Since we're using MongoDB, storing whole user objects in an array embedded in an another object is totally doable. If user objects didn't have any other relations with any other objects then this could be a good choice, but since user's are related to many other things and itself, then storing the records in another collection might make more sense, so we'll skip this option.

option (c) is the most versatile since neither the user or follows needs to know anything about each other and instead a mapping table keeps track of which user follows which user. Users and follow map can be created and retrieved independently each other. The only challenge is since we're using Mongo the joining of the three collections will consist of several data access requests through mongoose's ***populate*** function. If this were a relational

database then this option would be a top pick. We're going to settle on ***option (c)*** for our implementation and the files are listed below.

Use Case	***Method /resource/path***	***Request***	***Response Body***
User ***follows*** another user	POST /users/:uid/follows/:uid	User's and following user's PK	New following JSON
User ***unfollows*** another user	DELETE /users/:uid/unfollows/:uid		Delete status
User ***views*** a list of other users they are following	GET /users/:uid/userFollowing	User's PK	User following JSON array
User ***views*** a list of other users that are following them	GET /users/:uid /userFollowedBy	User's PK	User followers JSON array
User ***retrieves*** a following user	GET /users/:uid/following/:uid	User's and following user's PK	User JSON array
User ***retrieves*** a follower user	GET /users/:uid/followers/:uid	User's and follower user's PK	User JSON array

Design the Many to Many Bookmarks RESTful Web Service API (10pts)

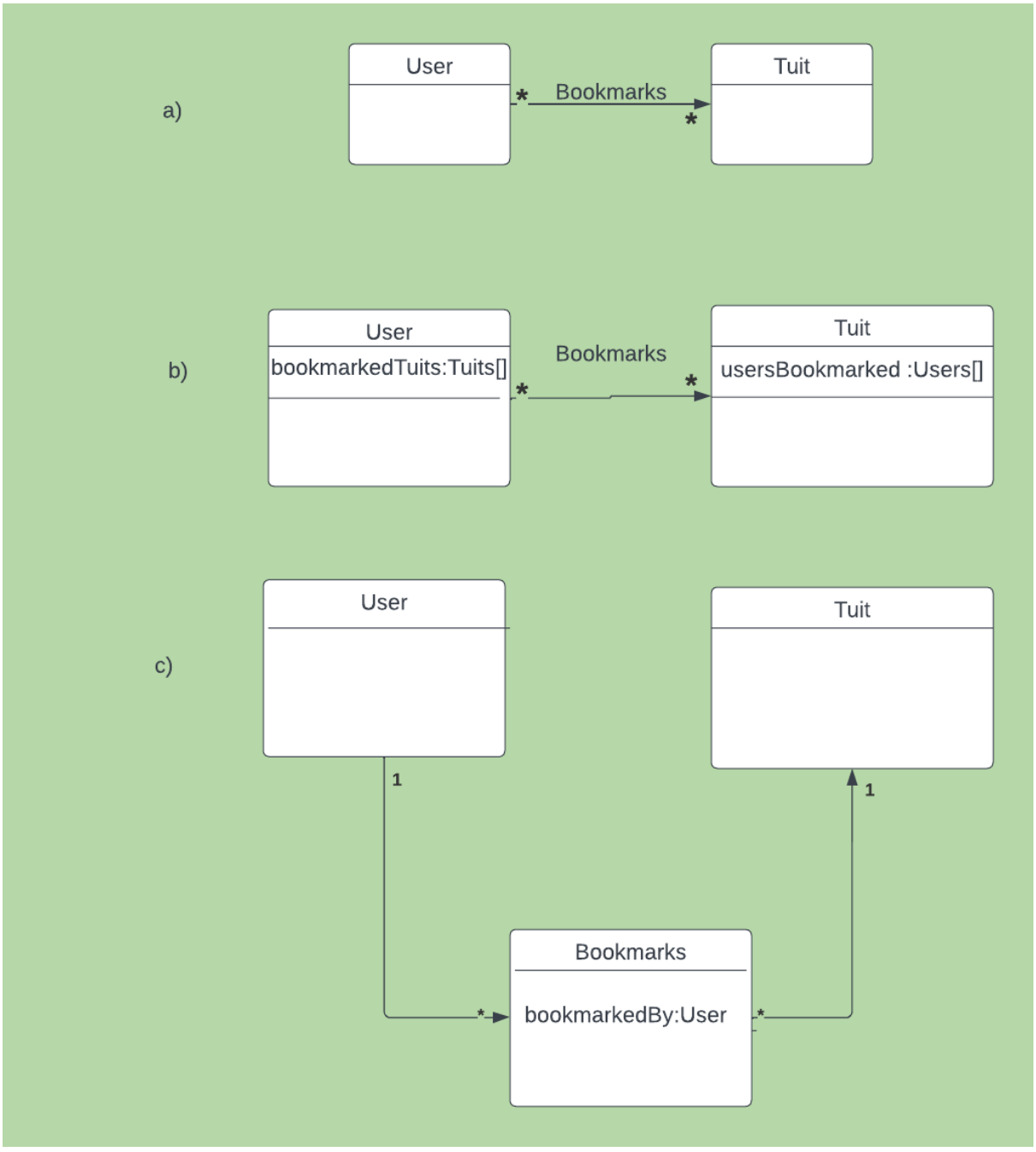
- A ***user*** can ***bookmark*** a tuit.
- A ***user*** can ***unbookmark*** a tuit
- A ***user views*** a list of tuits they have bookmarked
- ***Retrieve*** a list of users that bookmarked a tuit
- A ***user retrieves*** a bookmarked tuit

Nouns	***Verbs***	Verb phrases
a bookmark many bookmarks	***to bookmark, bookmarks, bookmarked***	user bookmarks a tuit

Identifying CRUD operations

Create	***Retrieve/Read***	***Update***	***Delete***

users bookmarks tuits ***users view bookmarks*** ***users unbookmark users***



Option (a) doesn't help since It just declares that there's a many to many relationship between users and tuits. We know that already.
Option (b) suggests that users could keep track of all the bookmarked tuits and tuits can keep track of all the users that bookmarked the tuit. This would be impossible to implement in a relational database, but doable in an OO language and in Mongo. The difficulty would be making sure that the arrays are consistent on both sides. That is, it would be an inconsistency that user1 bookmarks tuit2, but user1 would not be in tuit2's array.
Option (c) uses a mapping table to keep track of which user bookmarks which tuit. Again, this option is the most versatile, but might complicate joining in a non-

relational database, but this complexity is less compared to maintaining synchronous arrays in ***option (b)***. Also ***option (b)*** would require a transaction to make two updates atomically. So we're going to settle on ***option (c)*** to implement the bookmarks Web service API.

Use Case	***Method /resource/path***	***Request***	***Response Body***
User bookmarks a tuit	POST /users/:uid/bookmarks/:tid	User's and tuit's PK	New bookmark JSON
User unbookmarks a tuit	DELETE /users/:uid/bookmarks/:tid	User's and tuit's PK	Delete status
User views a list of tuits they have bookmarked	GET /users/:uid /bookmarks	User's PK	User bookmarks JSON array
User retrieves a bookmarked tuit	GET /users/:uid/bookmarked/:tid	User's and bookmarked tuit's PK	Tuit JSON array
User retrieves users that bookmarked a tuit	GET /tuits/:tid/bookmarked	Tuit's PK	Users JSON array

Design the Many to Many Messages RESTful Web Service API (10pts)

- User sends a message to another user
- User views a list of messages they have sent
- User views a list of messages sent to them
- User deletes a message
- User views a list of users messaging them
- User deletes a user from messages

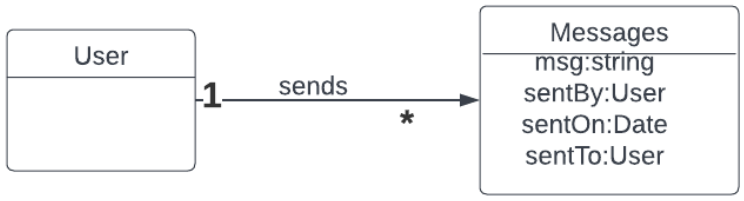
Nouns	***Verbs***	Verb phrases
a message many messages	***to message, messages, messaged***	user messages a user, user sends a message to a user

Identifying CRUD operations

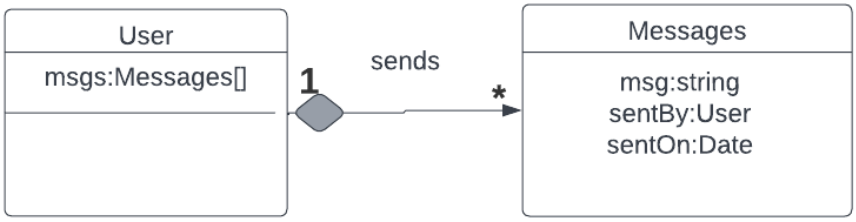
Create	***Retrieve/Read***	***Update***	***Delete***
users send messages	***users list messages***		***User deletes messages***
	user lists users in messages		***User deletes user from messages***

Analysis

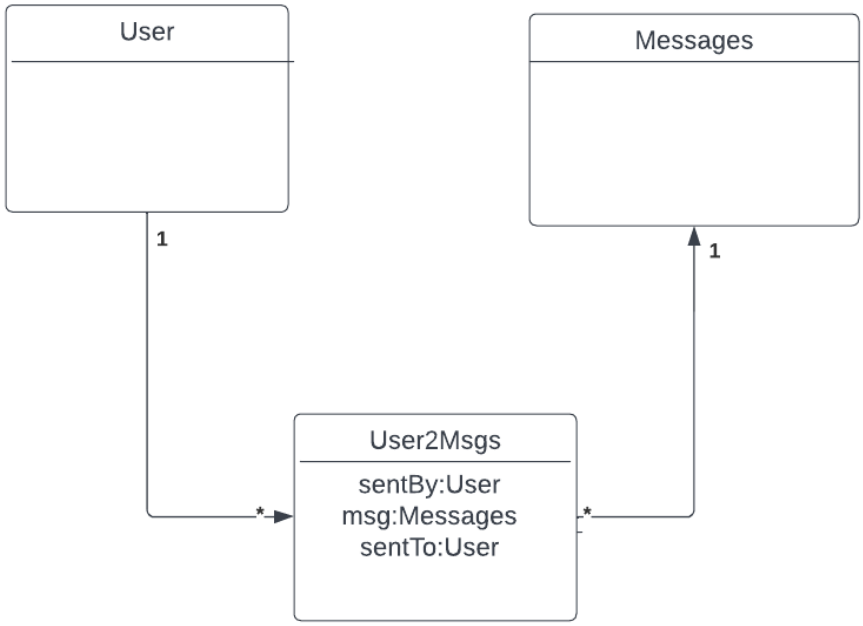
a)



b)



c)



Option (a) doesn't help. It just declares that there's a one to many relationship between users and messages. We know that already. ***Option (b)*** suggests that we could store the messages in an array in the sender's user. This option is not clear on whether we would store an actual object in the array or merely references to the message objects stored in some other collection. If we'd be using a relational database, this option would be impossible to implement. If this is an object oriented implementation then the array would clearly contain pointers to instances. Since we're using MongoDB, storing whole message objects in an array embedded in an another object is totally doable. If message objects didn't have any other relations with any other objects then this could be a good choice, but

since messages are related to many other things then storing the records in another collection might make more sense, so we'll skip this option. ***option (e)*** is the most versatile since neither the user or message needs to know anything about each other and instead a mapping table keeps track of which user sent which message and to whom. Users and messages can be created and retrieved independently each other. The only challenge is since we're using Mongo the joining of the three collections will consist of several data access requests through mongoose's ***populate*** function. If this were a relational database then this option would be a top pick. We're going to settle on ***option (c)*** for our implementation and the files are listed below.

Use Case	***Method /resource/path***	***Request***	***Response Body***
User sends a message another user	POST /users/:uid/messages/:mid	User's and message's PK	New message JSON
User deletes a ***message***	DELETE /users/:uid/messages/:mid	User's and message's PK	Delete status
User views a list of messages they have sent	GET /users/:uid/messages	User's PK	User messages JSON array
User views a list of messages sent to them	GET /users/:uid/ messages	User's PK	User messages JSON array