CS496 Assignment 3 Part 2
Robert Brauer, brauerr

Results of python tests:

Testing post users with bad username and password
[{"param":"email","msg":"Please Enter a Valid Email Address","value":"bademail"},{"param":"password","msg":"Please enter a valid password","value":""}]

Testing post userID with good username and password
581fcdcda57741be15574d38
[{"_id":"581fcdcda57741be15574d38","email":"tester@test.com","password":"password"}]

Getting tags
[{"_id":1,"text":"modern"},{"_id":2,"text":"silly"},{"_id":3,"text":"funny"},{"_id":4,"text":"sports"},{"_id":5,"text":"historical"},{"_id":6,"text":"inspirational"},{"_id":7,"text":"epic"}]

Post query for current users quotes
[]

Posting quote with tags outside index range
400
Tags don't exist for all ids in tags array

Posting quote with bad user id
400
User id does not exist

Posting good quote
{'rating': 5, 'user_id': '581fcdcda57741be15574d38', 'isFavorite': 'true', 'tags': [1, 2], 'text': 'this is just a test', 'said_by': 'a tester'}
{"text":"this is just a test","said_by":"a tester","isFavorite":"true","rating":5,"tags":[1,2],"user_id":"581fcdcda57741be15574d38","_id":"581fdb62f203fc3b293f9ea1"}

Updated quotes list:
[{"_id":"581fdb62f203fc3b293f9ea1","text":"this is just a test","said_by":"a tester","isFavorite":"true","rating":5,"tags":[1,2],"user_id":"581fcdcda57741be15574d38"}]

Updating quote content

Update response: <Response [200]>, [{"_id":"581fdb62f203fc3b293f9ea1","text":"this is some updated text","said_by":"a tester","isFavorite":"true","tags":[1,2],"user_id":"581fcdcda57741be15574d38"}]

Updated quotes list:
[{"_id":"581fdb62f203fc3b293f9ea1","text":"this is some updated text","said_by":"a tester","isFavorite":"true","tags":[1,2],"user_id":"581fcdcda57741be15574d38"}]

Deleting quote
Delete response: <Response [200]>

Quotes list after delete:
[]

An explanation of the URL structure used to access resources

Base URL:
http://104.236.251.255:3000/api

REST URLs:

GET /tags
Returns json of all tag objects usable for creating or updating quotes

POST /userID
Request containing json with “email” and “password” of valid user returns their id

GET /quotes
Returns json of all quote objects

POST /quotes/query
Request body containing json with any valid keys for a quote document returns any quotes matching all the query parameters.

POST /quotes
Request body containing json with at least a valid user_id, text, and said_by creates a new quote, so long as user_id and any ids in the tags array actually exist

PUT /quotes
Same as POST quote but additionally requires the “_id” key to identify an existing quote to update

DELETE /quotes
Request body containing “_id” of quote to be deleted

A description of which RESTful constraints you met and which you did not (you do not need to make a RESTful app, but you do need to know why it does not meet all the constraints of a RESTful API)

Client-Server: All processing done on the server side, client/consumer forms a proper request, and gets back a descriptive error response, or a completely formed success response.

Stateless: Everything is accomplished in a single call for each API operation - no sessions or state management is required to use it.

Layered System: This API is pretty simple, so there aren't actually any middleware calls to other services, but if they were, they'd be encapsulated inside the node.js methods on the server, and not something the consumer/client/requestor would have to manage.

Cacheable: No cache control metadata is currently explicitly updated and included in responses

Uniform interface: Standard set of directory structure-like URLs are used for API endpoints. JSON format is used for all requests and response data, which mirrors the data structure of the MongoDB backend.

A discussion of any changes you needed to make to the schema from last week

The biggest change that I needed to make was to create the tags collection and update the quotes collection to store an array of ids as references to tag documents, rather than storing the tag text directly in the tag array (embedded). Cleaning up old quotes proved to be a bit of work - would be a challenge to do on a database with a lot of records.

Having finished the API, what you would have done differently

There were a lot of things that I was learning as I put it together. One of the bigger challenges, not having done as much coding in JS, was dealing with asynchronous calls, and implementing methods that have to check certain collections and validate before querying other collections and adding, updating records etc. My example was checking whether a user_id posted to create or update a quote actually corresponds to a real user document, and whether an array of tag ids actually each correspond to a real tag document, before creating or updating the quote. I think I would explore using a library to help with some of these validations, Mongoose looks interesting, but by the time I discovered some documentation referencing it, I was very close to done implementing my post and put methods.