Lab 1

Due Sep 19, 2022 by 11:59pm **Points** 100 **Submitting** a file upload
File Types zip **Available** after Sep 12, 2022 at 12am

CS-554 Lab 1

Reviewing API Development

In this 546 Review lab, we will build SITwitter - A simple Twitter type application (That actually allows you to edit your "Sweets")

For this lab, you will submit a web server with the supplied routes and middlewares.

Verb	Route	Description
GET	/sweets	Shows a paginated list of Sweets in the system. By default, it will show the first `50` Sweets in the collection. If a [querystring (http://expressjs.com/en/api.html#req.query)](http://expressjs.com/en/api.html#req.query) variable `?page=n` is provided, you will show the the next 50 Sweets for that page `n` So `page=2` will show Sweets 51-100, `page=3` will show Sweets 101-150, `page=4` will show Sweets 151-200 and so on.. `page=1` would show the initial Sweets of 1-50 that you show by default on this route. If there are no Sweets for a page number (meaning there are no more Sweets in the DB, then you will return a 404 status code and message stating there are no more Sweets) **Hint: You can use the skip and limit cursors in MongoDB that we learned about in 546 to make this work.**
GET	/sweets/:id	Shows the Sweet with the supplied ID if a Sweet cannot be found for that ID, you will return a 404 Status code.
POST	/sweets	Creates a Sweet with the supplied detail and returns created object; fails request if not all details supplied. The user **MUST** be logged in to post a Sweet. In the request body, you will only be sending the `sweetText` and `sweetMood` fields of the Sweet. The `userThatPosted` will be populated from the currently logged in user (when they login, you will save a representation of the user in the session). You will initialize replies and the favorites as an empty array in your DB create function as there cannot be any replies or favorites on a Sweet, before the Sweet has been created. For `sweetMood` you will ONLY allow the following moods, if a mood not on this list is supplied, you will display an invalid mood

Verb	Route	Description
		error: Happy Sad Angry Excited Surprised Loved Blessed Greatful Blissful Silly Chill Motivated Emotional Annoyed Lucky Determined Bored Hungry Disappointed Worried
PATCH	/sweets/:id	Updates the Sweet with the supplied ID and returns the updated Sweet object; **Note**: **PATCH calls can have one or more fields in the request body!** **Note:** you cannot manipulate replies or favorites in this route! A user has to be logged in to update a Sweet **AND** they must be the same user who originally posted the Sweet. So if user A posts a Sweet, user B should NOT be able to update that Sweet. In the request body, you can only send the `sweetText` and the `sweetMood` (at least one of them needs to be supplied in the PATCH route, but both can also be present) fields of the Sweet.
POST	/sweets/:id/replies	Adds a new reply to the Sweet; ids must be generated by the server, and not supplied, a user needs to be logged in to post a reply.
DELETE	/sweets/:sweetId/:replyId	Deletes the reply with an id of `replyId` on the Sweet with an id of `sweetId` a user has to be logged in to delete a reply **AND** they must be the same user who originally posted the reply. So if user A posts a reply, user B should NOT be able to delete that reply.
POST	/sweets/:id/likes	Allows a user to like a Sweet, a user needs to be logged in to like a Sweet. If they have not already liked it, you will add the user's ID to the likes array in the Sweets document, if they have already liked

Verb	Route	Description
		the sweet and hit this route again, it should remove their ID from the likes array in the Sweets document
POST	/sweets/signup	Creates a new user in the system with the supplied detail and returns the created user document (sans password); fails request if not all details supplied.
POST	/sweets/login	Logs in a user with the supplied username and password. Returns the logged in user document (sans password). You will set the session so once they successfully log in, they will remain logged in until the session expires or they logout. You will store some way to identify the user in the session. You will store their `username` and their `_id` which will be read when they try to create a Sweet, try to update a Sweet (making sure they can only update a Sweet they originally posted), post a reply or delete a reply (making sure they can only delete a reply they posted), and adding/removing a Sweet favorite
GET	/sweets/logout	This route will expire/delete the `cookie/session` and inform the user that they have been logged out.

All PUT, POST, and PATCH routes expect their content to be in JSON format, supplied in the body.

All routes will return JSON.

Middleware

You will write and apply the following middlewares:

1. You will apply a middleware that will be applied to the POST, PUT and PATCH routes for the /sweets endpoint that will check if there is a logged in user, if there is not a user logged in, you will respond with the proper status code. For PUT and PATCH routes you also need to make sure the currently logged in user is the user who posted the Sweet that is being updated.
2. You will apply a middleware that will be applied to the POST and DELETE routes for the /sweets/:id/replies and /sweets/:sweetsId/:replyId endpoints respectively that will check if there is a logged in user, if there is not a user logged in, you will respond with the proper status code. For the DELETE route, you also need to make sure the currently logged in user is the user who posted the reply that is being deleted.

Database

You will use a module to abstract out the database calls.

You may find it helpful to reference the following 546 lecture code: **Lecture 4** (https://github.com/stevens-cs546-cs554/CS-546/tree/master/lecture_04/code) , **Lecture 5** (https://github.com/stevens-cs546-cs554/CS-546/tree/master/lecture_05/code) , **Lecture 6** (https://github.com/stevens-cs546-cs554/CS-546/tree/master/lecture_06/code) , **Lecture 10** (https://github.com/stevens-cs546-cs554/CS-546/tree/master/lecture_10/code)

You will store all data in a database named as such: `LastName-FirstName-CS554-Lab1`.

You may name the collection however you would like.

All ids must be generated by the server and be sufficiently random!

The Sweets document

```
{
  _id: new ObjectID(),
  "sweetText": string,
  "sweetMood": string,
  "userThatPosted": {_id: ObjectID, username: string},
  "replies": [objects],
  "likes": [of user ids that have favorited the Sweet]
}
```

The reply object (stored as a sub-document in the Sweet document)

```
{
  _id: new ObjectID(),
  "userThatPostedReply": {_id:ObjectID, username: string},
  "reply": string
}
```

The user document: You will use bcrypt to hash the password to store in the DB for signup and for login you will use the compare method to validate the correct password

```
{
  _id: new ObjectID(),
  "name": string,
  "username": string,
  "password": hashedPW
}
```

Example Sweet:

```
{
  "_id": "61294dadd90ffc066cd03bed",
  "sweetText": "I am ready to learn React!",
  "sweetMood": "Excited",
  "userThatPosted": {_id: "61294dadd90ffc066cd03bee", username: "graffixnyc"},
  "replies": [
      {
        "_id": "61294dadd90ffc066cd03bef",
        "userThatPostedReply": {_id:"51294dadd90ffc066cd03bff", username: "ZeroCool"},
        "reply": "Me either! React will be awesome!"
      },
      {
        "_id": "61296014082fe5073f9ba4f2",
        "userThatPostedReply": {_id:"6129617a082fe5073f9ba4f5", username: "progman716"}
```

```
          "reply": "I like Vue JS better than React"
        }
    ],
    "likes": ["51294dadd90ffc066cd03bff", "61296014082fe5073f9ba4f2"]
  }
```

Example User Document

```
{
  _id: "61294dadd90ffc066cd03bee",
"name": "Patrick Hill", "username":"graffixnyc",
"password": "$2a$16$7JKSiEmoP3GNDSalogqgPu0sUbwder7CAN/5wnvCWe6xCKAKwlTD."
}
```

Error Checking

1. **You must error check all routes checking correct data types, making sure all the input is there, in the correct range etc..**
2. **You must error check all DB functions checking correct data types, making sure all the input is there, in the correct range etc..**
3. **You must fail with proper and valid HTTP status codes depending on the failure type**
4. **Do not forget to check for proper datatypes in the query string parameters for page (it should be a positive number, if is not a positive number, you should throw an error)**

Notes

1. Remember to submit your `package.json` file but **not** your `node_modules` folder.