Lab 2

Due Feb 24 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip

CS-554 Lab 2

Redis

For this assignment, you will practice your usage of redis and async / await together.

You will use your lab 1 code for this assignment and will implement caching.

Your cache

You will cache your data in Redis for this assignment after it has been accessed; you should not cache all data by default. When a request is made to one of the routes, you will first check to see if it is already stored in the cache. If it is, you will serve it from the cache. If it's not stored in the cache, you will then query the DB to get the data, store it in the cache for the next time it is accessed and will respond with the JSON data returned from the DB.

For the most accessed recipes, you will need to research **sorted sets** in Redis (these are sometimes referred to as scoreboards or leaderboards) We will be storing the the entire recipe object and then a count of how many times that recipe was accessed. If someone accessed a recipe that has been accessed already, you will increment the count. If the access a recipe not in the list, you will add it to the list and set the count initially to 1.

Your Routes

You will be using the same routes as you did in lab 1, with the same constraints as lab 1. You also will implement one new route. The original routes are displayed below from lab 1. I added in the caching for each route.

Verb	Route	Description
GET	/recipes	Shows a paginated list of recipes in the system. By default, it will show first `50` recipes in the collection. If a **querystring (http://expressjs.com/en/api.html#req.query)** variable `?page=n` is provi you will show the the next 50 recipes for that page `n` So `page=2` wil recipes 51-100, `page=3` will show recipes 101-150, `page=4` will show recipes 151-200 and so on.. `page=1` would show the initial recipes of

Verb	Route	Description
		that you show by default on this route. If there are no recipes for a pag number (meaning there are no more recipes in the DB, then you will re 404 status code and message stating there are no more recipes) **Hint: can use the skip and limit cursors in MongoDB that we learned ab 546 to make this work.** **REDIS IMPLEMENTATION: WHEN THIS ROUTE IS ACCESSED, YO WILL CREATE A MIDDLEWARE THAT CHECKS TO SEE IF THE RESULTS FOR THAT PAGE ARE IN THE CACHE, IF THEY ARE IN T CACHE, YOU WILL RESPOND WITH THE JSON DATA THAT IS IN T CACHE, IF IT'S NOT IN THE CACHE, IT WILL 1. QUERY THE DB FO THE DATA, 2. STORE THE DATA IN CACHE, 3. RESPOND WITH TH DATA RETURNED FROM THE DB. FOR THE ROOT ROUTE, YOU C STORE THIS AS PAGE 1 SINCE PAGE=1 RETURNS THE SAME RESULTS AS THE ROOT ROUTE.**
GET	/recipes/:id	Shows the recipe with the supplied ID. if a recipe cannot be found for tl you will return a 404 Status code along with an error message. **REDIS IMPLEMENTATION: WHEN THIS ROUTE IS ACCESSED, YO WILL CREATE A MIDDLEWARE THAT CHECKS TO SEE IF THE RE BEING REQUESTED IS IN THE CACHE, IF IT IS IN THE CACHE, YO WILL RESPOND WITH THE JSON DATA THAT IS IN THE CACHE, II NOT IN THE CACHE, IT WILL 1. QUERY THE DB FOR THE DATA, 2 STORE THE DATA IN CACHE, 3. RESPOND WITH THE DATA RETU FROM THE DB.** *****In addition to caching each recipe, you will also have to keep tr the number of times each recipe is requested in a sorted list with You will store the entire recipe object in the list and then have a fi that keeps track of how many times that recipe was accessed. Th sorted list will be accessed from the /mostaccessed route. See the /mostaccessed for more specific instructions.**
POST	/recipes	Creates a recipe with the supplied detail and returns created object; fai request if not all details supplied. The user **MUST** be logged in to post recipe. In the request body, you will not be sending the `userThatPost` will be populated from the currently logged in user (when they login, yo save a representation of the user in the session). **You will initialize comments and the likes as empty arrays in your DB create functic there cannot be any comments or likes on a recipe, before the rec has been created.**

Verb	Route	Description
		Please see example schema below. The fields in the request body will the following constraints: `title` : Must be a valid string. No strings with empty spaces allowed data must make sense for the data you are storing. `ingredients` : Each element must be a valid string, no strings with er spaces and there should be at least 3 valid string elements in the array data must make sense for the data you are storing. The minimum char for each ingredient should be 3 characters and the max 50 characters. `steps` : Each element must be a valid string, no strings with empty sp and there should be at least 5 valid string elements in the array. The da must make sense for the data you are storing. The minimum number o characters should be 20. No max character constraint. `cookingSkillRequired` you will ONLY allow the following values, if a not on this list is supplied, you will display an invalid cooking skill requir error: "Novice", "Intermediate", "Advanced" This route will return the newly created recipe once it's inserted into the database. **REDIS IMPLEMENTATION: WHEN A NEW RECIPE IS CREATED, YO WILL STORE IT IN THE CACHE AS YOU DID FOR RECIPES IN THE /recipes/:id ROUTE AND RESPOND WITH THE JSON DATA FROM DB AFTER THE RECIPE IS CREATED. Don't forget to update the s list, to include this new recipe as one of the ones that was access Since the recipe is being created for the first time in the route, the has been accessed 1 time after it's created**
PATCH	/recipes/:id	Updates the recipe with the supplied ID and returns the updated recipe object; **Note**: **PATCH calls can have one or more fields in the reque body!** **Note:** you cannot manipulate comments, likes or modify the `userThatPosted` object in this route! A user has to be logged in to up recipe **AND** they must be the same user who originally posted the recip if user A posts a recipe, user B should NOT be able to update that recip The constraints for the fields are the same as they are in the POST rou least one field needs to be supplied in the request body and the value r be different than what is currently stored in the DB, but more than one can also be present. This route will return the newly updated recipe.

Verb	Route	Description
		REDIS IMPLEMENTATION: WHEN A RECIPE IS UPDATED, IF THAT RECIPE ID IS STORED IN THE CACHE, YOU WILL NEED TO UPDA THE CACHE WITH THE UPDATED DATA. YOU WILL STORE IT IN T CACHE AS YOU DID FOR RECIPES IN THE /recipes/:id ROUTE AN RESPOND WITH THE JSON DATA FROM THE DB AFTER THE REC UPDATED. Don't forget to update the sorted list, to include this u recipe data as one of the ones that was accessed.
POST	/recipes/:id/comments	Adds a new comment to the recipe; ids must be generated by the serv not supplied. a user needs to be logged in to post a comment. This route will return the entire recipe object, showing the new commer the data. **REDIS IMPLEMENTATION: WHEN A COMMENT IS POSTED, IF THA RECIPE ID IS STORED IN THE CACHE, YOU WILL NEED TO UPDA THE CACHE WITH THE UPDATED DATA FOR THAT RECIPE. YOU STORE IT IN THE CACHE AS YOU DID FOR RECIPES IN THE /reci ROUTE AND RESPOND WITH THE JSON DATA FROM THE DB AF THE RECIPE IS UPDATED. Don't forget to update the sorted list, t include this updated recipe data as one of the ones that was acce**
DELETE	/recipes/:recipeId/:commentId	Deletes the comment with an id of `commentId` on the recipe with an i `recipeId`. A user has to be logged in to delete a comment **AND** they be the same user who originally posted the comment. So if user A pos comment, user B should NOT be able to delete that comment. This route will return the entire recipe object, showing the deleted com gone from the data. **REDIS IMPLEMENTATION: WHEN A COMMENT IS DELETED, IF TH RECIPE ID IS STORED IN THE CACHE, YOU WILL NEED TO UPDA THE CACHE WITH THE UPDATED DATA FOR THAT RECIPE. YOU STORE IT IN THE CACHE AS YOU DID FOR RECIPES IN THE /reci ROUTE AND RESPOND WITH THE JSON DATA FROM THE DB AF THE RECIPE IS UPDATED. Don't forget to update the sorted list, t include this updated recipe data as one of the ones that was acce**
POST	/recipes/:id/likes	Allows a user to like a recipe. A user needs to be logged in to like a rec they have not already liked it, you will add the user's ID to the likes arra the recipe document. If they have already liked the recipe and hit this r again, it should remove their ID from the likes array in the recipe docur

Verb	Route	Description
		This route will return the entire recipe object, showing the new like in th (if they liked it, or showing their id is removed if they deleted it). **REDIS IMPLEMENTATION: WHEN A LIKE IS POSTED, IF THAT RE ID IS STORED IN THE CACHE, YOU WILL NEED TO UPDATE THE CACHE WITH THE UPDATED DATA FOR THAT RECIPE. YOU WILL STORE IT IN THE CACHE AS YOU DID FOR RECIPES IN THE /reci ROUTE AND RESPOND WITH THE JSON DATA FROM THE DB AF THE RECIPE IS UPDATED. Don't forget to update the sorted list, t include this updated recipe data as one of the ones that was acce**
POST	/signup	Creates a new user in the system with the supplied detail and returns t created user document (sans password); fails request if not all details supplied. The username must be alphanumeric and at least 3 characte long. The password should be 6 characters minimum, with at least one lowercase letter, one uppercase letter, one number and one special ch contained in it.
POST	/login	Logs in a user with the supplied username and password. Returns the in user document (sans password). You will set the session so once th successfully log in, they will remain logged in until the session expires logout. You will store some way to identify the user in the session. You store their `username` and their `_id` which will be read when they tr create a recipe, try to update a recipe (making sure they can only upda recipe they originally posted), post a comment or delete a comment (n sure they can only delete a reply they posted), and adding/removing a like.
GET	/logout	This route will expire/delete the `cookie/session` and inform the user they have been logged out.
GET	/mostaccessed	This route will display the top 10 most accessed recipes that are storec sorted list with Redis. When a recipe is requested from the GET recipes/:id route, you will ad your cached sorted list in Redis to count how many times that specific has been accessed. You should either set the count to 1 if it has not be accessed, or increment it +1 if it has already been accessed before. **So, GET /mostaccessed will return the top 10 recipes from the sor list, with the recipe that is most accessed being first in the list. Yo return ALL the recipe data for each item in the list. (You should ha stored this in the list in the recipes/:id route** **Please see how I do something similar for the top most searched in the lecture code example here (https://github.com/stevens-cs5**

Verb	Route	Description
		cs554/CS-554/tree/master/redis/redis_full_page_caching_handlebars_new_redis_pa

Middleware

You will write and apply the following middleware functions:

1. You will apply a middleware that will be applied to the POST, PUT and PATCH routes for the /recipes 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 and display and error message. (A non-logged in user SHOULD be able to access the GET /recipes route)
2. You will apply a middleware that will be applied to POST and DELETE for the /recipes/:id/comments and /recipes/:recipeId/:commentId 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 and display and error message.
3. The third middleware will apply to the entire application and will log all request bodies if there is a request body (GET routes can/will just log an empty object for the request body). Do not log passwords from the request body if the request body contains a password field. You will also log the url path they are requesting, and the HTTP verb they are using to make the request.
4. The fourth will apply to the entire application and will keep track of many times a particular URL has been requested, updating and logging with each request.

Extra Credit:

For extra credit, instead of storing the session in Express-Session, you will store the session in Redis. If we restart your server, It should read the session in Redis and the session should still be valid. If you implement full session caching and it works properly, you will get +10 for extra credit.

General Requirements

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