Peer-graded Assignment: User Authentication

Submit by December 2, 11:59 PM PST

Important Information
It is especially important to submit this assignment before the deadline, December 2, 11:59 PM PST, because it must be graded by others. If you submit late, there may not be enough classmates around to review your work. This makes it difficult - and in some cases, impossible - to produce a grade. Submit on time to avoid these risks.

Instructions

My submission

Discussions

In this assignment you will continue the exploration of user authentication. We have already set up the REST API server to validate an ordinary user. Now, you will extend this to verify an Admin and grant appropriate privileges to an Admin. In addition you will allow only a registered user to update and delete his/her submitted comments. Neither another user, nor an Admin can edit these comments.

Step-By-Step Assignment Instructions less

Assignment Overview

At the end of this assignment, you would have completed the following:

- Check if a verified ordinary user also has Admin privileges.
- Allow any one to perform GET operations
- Allow only an Admin to perform POST, PUT and DELETE operations
- Allow an Admin to be able to GET all the registered users' information from the database
- Allow a registered user to submit comments (already completed), update a submitted comment and delete a submitted comment. The user should be restricted to perform such operations only on his/her own comments. No user or even the Admin can edit or delete the comments submitted by other users.

Assignment Requirements

This assignment is divided into three tasks as detailed below:

Task 1

In this task you will implement a new function named *verifyAdmin()* in *authenticate.js* file. This function will check an ordinary user to see if s/he has Admin privileges. In order to perform this check, note that all users have an additional field stored in their records named *admin*, that is a boolean flag, set to *false* by default. Furthermore, when the user's token is checked in *verifyOrdinaryUser()* function, it will load a new property named *user* to the *request* object. This will be available to you if the *verifyAdmin()* follows *verifyUser()* in the middleware order in Express. From this *req* object, you can obtain the admin flag of the user's information by using the following expression:

```
req.user.admin
```

You can use this to decide if the user is an administrator. The *verifyAdmin()* function will call *next();* if the user is an Admin, otherwise it will return next(err); If an ordinary user performs this operation, you should return an error by calling next(err) with the status of 403, and a message "You are not authorized to perform this operation!".

Note: See the video on how to set up an Admin account

Task2

In this task you will update all the routes in the REST API to ensure that only the Admins can perform POST, PUT and DELETE operations. Update the code for all the routers to support this. These operations should be supported for the following end points:

- POST, PUT and DELETE operations on /dishes and /dishes/:dishId
- DELETE operation on /dishes/:dishId/comments
- POST, PUT and DELETE operations on /promotions and /promotions/:promoId
- POST, PUT and DELETE operations on /leaders and /leaders/:leaderId

Task 3

In this task you will now activate the /users REST API end point. When an Admin sends a GET request to http://localhost:3000/users you will return the details of all the users. Ordinary users are forbidden from performing this operation.

Task 4

In this task you will allow a registered user to update or delete his/her own comment. Recall that the comment already stores the author's ID. When a user performs a PUT or DELETE operation on the /dishes/:dishId/comments/:commentId REST API end point, you will check to ensure that the user performing the operation is the same as the user that submitted the comment. You will allow the operation to be performed only if the user's ID matches the id of the comment's author. Note that the User's ID is available from the *req.user* property of the *req* object. Also ObjectIDs behave like Strings, and hence when comparing two ObjectIDs, you should use the *Id1.equals(id2)* syntax.

Review criteria less

Your assignment will be graded based on the following criteria:

Task 1

- You have implemented the *verifyAdmin()* function in *authenticate.js*.
- The *verifyAdmin()* function will allow you to proceed forward along the normal path of middleware execution if you are an Admin
- The *verifyAdmin()* function will prevent you from proceeding further if you do not have Admin privileges, and will send an error message to you in the reply.

Task 2

- Any one is restricted to perform only the GET operation on the resources/REST API end points.
- An Admin (who must be first checked to make sure is an ordinary user), can perform the GET, PUT, POST and DELETE operations on any of the resources/ REST API end points.

Task 3

- A GET operation on http://localhost:3000/users by an Admin will return the details of the registered users
- An ordinary user (without Admin privileges) cannot perform the GET operation on http://localhost:3000/users.

Task 4

- A registered user is allowed to update and delete his/her own comments.
- Any user or an Admin cannot update or delete the comment posted by other users.