Lab 4: Handling Authentication

Learning Objectives

- Implement authentication for your assignment application

This week's exercises

To complete this week's exercises, make sure that you have first completed Exercise 4 from last week's lab sheet. You will need some assignment routes implemented so that you can test that the authentication endpoints work.

If you don't understand any of the exercises, go back over last week's content. Everything here builds on the previous labs.

Important: Remember that this is now assignment work. Plagiarism rules apply and the code you write MUST be your own!

Exercise 1: Understanding the Authentication workflow

Before continuing, make sure that you understand the authentication workflow for the assignment:

1. Recap this week's lecture
2. Go back to the Spacebook server we used in week 1:
   a. Can you create a user?
   b. Can you log in as that user?
   c. Can you hit an endpoint that requires authentication?
   d. Can you log out?

Exercise 2: Create the model functions

First, create the model functions that will be used for authentication. The below table lists each of the functions you will need and describes what they do. These functions should go in a new user models file.

* There is a full solution in the lecture slides
** There is a partial solution in the lecture slides

Function name	Visibility	Parameters	Description
getHash*	Private	Password, salt	Creates and returns a hash of the password and salt as a Hex string
addNewUser**	Public	User, done	1) creates a random salt and a hash of the password and salt (using the getHash function) 2) Inserts all the relevant user data into the users table
authenticateUser*	Public	Email, password, done	1) Gets the user record from the DB using the email

			2) Passes the incoming password and salt from the DB to the getHash function 3) Checks the generated hash is the same as the DB hash 4) Handles all errors throughout – what if the email doesn't exist? What if the hashes don't match?
getToken	Public	Id, done	Gets the token from the user's token where the user's ID matches the parameter ID
setToken*	Public	Id, done	Creates a random token and adds it to the DB for the user with the given ID
removeToken*	Public	Token, done	Clears the provided token from the DB
getIDFromToken**	Public	Token, done	1) Checks the token is valid 2) Gets the user ID from the DB where the token matches the incoming token 3) Handles any errors

Exercise 3: Implement the POST /user endpoint

1. Create a routes file for the user endpoints
2. Create the POST /user route, instruct it to call a function in the controller
3. Create the controller file and the scaffold for the function you have just created
4. Validate the incoming data to ensure it matches the API specification
5. Create a user object with all the relevant data needed
6. Pass it to your model function. The first parameter will be the user object you have just created, and the second will be a callback function
7. The callback function should handle any errors. If there are no errors, then return a 201 response with the ID of the new user. (Note: You may need to edit your model function to ensure it passes the ID to the done() function
8. Test your endpoint

Exercise 4: Implement the POST /login endpoint*

1. Create the route and the scaffold function in the controller
2. Validate the incoming data against the API
3. Authenticate the user using the model function
4. In the callback:
    a. Handle any errors
    b. Attempt to get the token from the DB (if the user has already logged in and has a token, then you want to return the existing token)
    c. If no token exists, create a new one and return to the client
5. Test your endpoint

Exercise 5: Implement the POST /logout endpoint

1. Create the route and scaffold function in the controller
2. Get the token header from the request
3. Remove the token from the DB using the model function
4. Handle the response, check for errors
5. Test your endpoint

Exercise 6: Understanding middleware

Before going any further, make sure that you understand how middleware works. You can recap this week's lecture, but also check out this blog post: https://blog.webdevsimplified.com/2019-12/express-middleware-in-depth/

Exercise 7: Implement a middleware function*

1. In your app directory, create a directory called lib (for libraries)
2. Inside the lib directory, create a authentication.js file
3. In this file, create a middleware function. The function needs to:
    a. Get the X-Authorization token
    b. Use the model functions to convert it to an ID
    c. If there is no ID returned, then the middleware function should return a 401 Unauthorised response to the client
    d. Else, execute next() to keep processing
4. Import your middleware function into your routes files and add it to the route of every endpoint which requires authentication
5. Test your endpoints

Assignment: Back-end

You now have all the tools necessary for completing the rest of the assignment backend. The next two weeks of labs will not have any lab sheet. Instead, you will be expected to finish off your back-end ready for us to start looking at the front-end application.

Make sure you understand and test every endpoint and run the tests provided for you (as that is how you will be marked)

Also, make sure you go back over the end-points created last week to embed the authentication elements properly. For example, last week we hardcoded the "created_by" field when adding a new article. Now you can update that function so instead it gets the ID from the token using the getIDFromToken function that you have created today.