Lab 04-01 100/100 Great Job! Please see suggestion below

horizontal lineOne suggestion is to use an Error Handler so that when isLoggedIn returns false that the returned output is not a complete HTML page, but rather a message like this

{

"error": "Unauthorized. Please login first."

}

// Error handler

router.use((error, req, res, next) => {

  if (!req.auth) {

    res.status(401).json({ error: "Unauthorized. Please login first." });

    return;

  }

  else {

    res.status(500).json({ error: error.message });

    return

  }

});

# Issue Tracker - Phase 4

This lab is a continuation of the Issue Tracker project that we started previously. In this assignment we will be implementing various security features, around the subject of Authentication.

This lab assignment will guide you through implementing the following requirements:

1. Users can register an account for themselves.
2. Users can log into the system using their **email address** and **password.**
3. Users can change their own password.
4. All passwords must be **hashed** before they are stored in the database. Plain text passwords will **never** be stored in the database.
5. When the user registers an account, or logs in, issue a JWT token to track their session. This token must be stored in a cookie, as well as returned from these two routes.
6. Use an application-wide middleware module **(@merlin/express-auth)** to validate the token, and to centralize authentication logic.
7. Only logged in users may access the API. *(But the register and login routes do not require a login.)*
8. All edits must be **tracked** with the **date & time** the change was made, the **user** who made the change, and **what** was changed.

# Step 1. Use your existing GitHub Repository

Use your **awd1111-issue-tracker** repository that you created earlier.

**Do not create a new repository!**

# Step 2. Install new dependencies

Install the following additional dependencies, using the **"npm install"** command:

|  |
| --- |
| npm install bcrypt cookie-parser jsonwebtoken |

# 

# Step 3. Update .env file

Update the .env file to include a JWT\_SECRET

# Exercise 1. Update Register Route (10pts)

Update the logic to use the things that we have learned:

1. Hash passwords

|  |
| --- |
| user.password = await bcrypt.hash(user.password, 10); |

1. Issue a new JWT token
2. Save the JWT token in a cookie
3. And send the token back in the response

|  |
| --- |
| res.json({ message: 'User Registered!', userId, authToken }); |

# 

# Exercise 2. Update Login Route (5pts)

Update the logic to use the things that we have learned:

1. Compare hashed passwords

|  |
| --- |
| if (user && await bcrypt.compare(password, user.password)) { |

1. Issue a new JWT token
2. Save the JWT token in a cookie
3. And send the token back in the response

|  |
| --- |
| res.json({ message: 'Welcome Back!', userId, authToken }); |

# Exercise 3. Update User API (30pts)

## POST /api/user/register (5pts)

* *Registration does not require a login, anonymous users can register an account.*
* When a new user is registered set the following on the **user** document:
  + \_id = *new id*
  + createdOn = *current date*
  + role = [‘developer’]
* Add a record to the **edits** collection to track the changes:
  + timestamp = *current date*
  + col = "user"
  + op = "insert"
  + target = { userId }
  + update = user

## POST /api/user/login --

* *If the user credentials are correct generate a JSON Web token and store it in a cookie.*
* *No other changes needed. You do not need to track edits.*

## GET /api/user/list (2.5pts)

* Send back a 401 error message, if the user is not logged in.

## GET /api/user/me --

* *Implement this new route for users to view their own account/profile.*
* Send back a 401 error message, if the user is not logged in.
* Otherwise query the database by userId, and return the data to the user.

## GET /api/user/:userId (2.5pts)

* Send back a 401 error message, if the user is not logged in.

## PUT /api/user/me (10pts)

* *Implement this new route for users to edit their own account/profile.*
* Send back a 401 error message, if the user is not logged in.
* Hash the password if it changes.
* Issue a new token with the updated information.
* If any fields will be updated, also update the following:
  + lastUpdatedOn = *current date*
  + lastUpdatedBy = *information pulled from req.auth*
* Add a record to the **edits** collection to track the changes:
  + timestamp = *current date*
  + col = "user"
  + op = "update"
  + target = { userId }
  + update = *fields that were changed and their new values*
  + auth = req.auth

## 

## PUT /api/user/:userId (5pts)

* *This route will be used for administrators to update the accounts/profiles of other users.*
* Send back a 401 error message if the user is not logged in.
* Hash the password if it changes.
* Issue a new token if the user is updating themselves.
* If any fields will be updated, also update the following:
  + lastUpdatedOn = *current date*
  + lastUpdatedBy = *information pulled from req.auth*
* Add a record to the **edits** collection to track the changes:
  + timestamp = *current date*
  + col = "user"
  + op = "update"
  + target = { userId }
  + update = *fields that were changed and their new values*
  + auth = req.auth

## DELETE /api/user/:userId (5pts)

* *This route will be used for administrators to delete accounts of spammers/trolls.*
* Send back a 401 error message if the user is not logged in.
* Add a record to the **edits** collection to track the changes:
  + timestamp = *current date*
  + col = "user"
  + op = "delete"
  + target = { userId }
  + auth = req.auth

# 

# Exercise 4. Update Bug API (30 pts)

## GET /api/bug/list (2.5pts)

* Send back a 401 error message if the user is not logged in.

## GET /api/bug/:bugId (2.5pts)

* Send back a 401 error message if the user is not logged in.

## PUT /api/bug/new (5pts)

* Send back a 401 error message if the user is not logged in.
* When the bug is created set the following on the **bug** document:
  + \_id = *new id*
  + createdOn = *current date*
  + createdBy = *information pulled from req.auth*
  + classification = "unclassified"
  + closed = false
* Add a record to the **edits** collection to track the changes:
  + timestamp = *current date*
  + col = "bug"
  + op = "insert"
  + target = { bugId }
  + update = bug
  + auth = req.auth

## PUT /api/bug/:bugId (5pts)

* Send back a 401 error message if the user is not logged in.
* If any fields will be updated, also update the following:
  + lastUpdatedOn = *current date*
  + lastUpdatedBy = *information pulled from req.auth*
* Add a record to the **edits** collection to track the changes:
  + timestamp = *current date*
  + col = "bug"
  + op = "update"
  + target = { bugId }
  + update = *fields that were changed and their new values*
  + auth = req.auth

## PUT /api/bug/:bugId/classify (5pts)

* Send back a 401 error message if the user is not logged in.
* Update the following fields on the **bug** document:
  + classifiedOn = *current date*
  + classifiedBy = *information pulled from req.auth*
* Add a record to the **edits** collection to track the changes:
  + timestamp = *current date*
  + col = "bug"
  + op = "update"
  + target = { bugId }
  + update = *fields that were changed and their new values*
  + auth = req.auth

## PUT /api/bug/:bugId/assign (5pts)

* Send back a 401 error message if the user is not logged in.
* Update the following fields on the **bug** document:
  + assignedOn = *current date*
  + assignedBy = *information pulled from req.auth*
* Add a record to the **edits** collection to track the changes:
  + *same as above*

## PUT /api/bug/:bugId/close (5pts)

* Send back a 401 error message if the user is not logged in.
* Update the following fields on the **bug** document:
  + closedOn *(set to null if bug is re-opened)*
  + closedBy *(set to null if bug is re-opened)*
* Add a record to the **edits** collection to track the changes:
  + *same as above*

# 

# Exercise 5. Update Comment API (10 pts)

GET /api/bug/:bugId/comment/list (2.5 pts)

* Send back a 401 error message if the user is not logged in.

GET /api/bug/:bugId/comment/:commentId (2.5 pts)

* Send back a 401 error message if the user is not logged in.

PUT /api/bug/:bugId/comment/new (5 pts)

* Send back a 401 error message if the user is not logged in.
* Use **req.auth** to determine who posted the comment, instead of reading from req.body.

# Exercise 6. Update Test Case API (15 pts)

GET /api/bug/:bugId/test/list (2.5 pts)

* Send back a 401 error message if the user is not logged in.

GET /api/bug/:bugId/test/:testId (2.5 pts)

* Send back a 401 error message if the user is not logged in.

PUT /api/bug/:bugId/test/new (2.5 pts)

* Send back a 401 error message if the user is not logged in.
* Set the **createdOn** and **createdBy** fields on the test case.
* Add a record to the edits collection to track changes.

PUT /api/bug/:bugId/test/:testId (2.5 pts)

* Send back a 401 error message if the user is not logged in.
* Set the **lastUpdatedOn** and **lastUpdatedBy** fields on the test case.
* Add a record to the edits collection to track changes.

DELETE /api/bug/:bugId/test/:testId (5 pts)

* Send back a 401 error message if the user is not logged in.
* Add a record to the edits collection to track changes.