Lauren Meeker

Module 5 Assignment: Assignment 1

Due: 03/19/23

Professor Dinesh Sthapit

Output(s)

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

JWT Codementor:Diagram

Description automatically generated

The functionality we need to implement or add to the current project is:[[1]](#footnote-1)

* To obtain functionality, we’ll need to adjust the userController.js by running touchapi/controllers/userController.js. We’ll create three separate handlers to handle ‘registration’, ‘sign in’, ‘login required’. The sign in handler will need to check if the user is already saved in the database (if yes, have to check if the parameters matches). In loginRequired, it checks to see if the user is signed in.
* Make sure jsonwebtoken is installed
  + Npm install jsonwebtoken 🡪 save
* Add two routes for user activities, we’ll have to make the handlers available within the routes by updating
  + Var todoList = require(‘../controllers/todoListController’);
    - To
      * Var todoList = require(‘../controllers/todoListController’),
      * userHandlers = require(‘../controllers/userController.js’);
  + add the loginRequired handler to the post request on /tasks
  + adds the route for the user to be able to sign in and have access to the app with the newly created credentials
* Add the new user we created previously and add a middleware to the express server that checks the state of the user (to the server.js file).
  + To add the user to server.js, we’ll have to add
    - Var User = require(‘./api/models/userModel’),
    - Jsonwebtoken = require(“jsonwebtoken”);

Additional validation you need to implement or add to your data model in this project[[2]](#footnote-2)

* Querying (finding documents/documents can be retrieved using a model’s find, findByld, findOne, or where static functions).
* Update method (each model has its own update method for modifying documents in the database without returning them to your application)
  + Tank.updateOne({size: ‘large’ }, { name: ‘T-90’ }, function(err, res) {
  + });
* Change streams (provides way for you to listen to inserts and updates going to the MongoDB)
* **async** **function** **run**() {
* *// Create a new mongoose model*
* **const** personSchema = **new** mongoose.**Schema**({
* name: **String**
* });
* **const** **Person** = mongoose.**model**('Person', personSchema);
* *// Create a change stream. The 'change' event gets emitted when there's a*
* *// change in the database*
* **Person**.**watch**().
* **on**('change', data => console.**log**(**new** **Date**(), data));
* *// Insert a doc, will trigger the change stream handler above*
* console.**log**(**new** **Date**(), 'Inserting doc');
* **await** **Person**.**create**({ name: 'Axl Rose' });
* }
* Views (read-only collections that contain data computed from other collections using aggregations). In the MongoDB you should define a separate model for each view(s), also possible to create a view using createCollection().
* *// Make sure to disable `autoCreate` and `autoIndex` for Views,*
* *// because you want to create the collection manually.*
* **const** userSchema = **new** **Schema**({
* name: **String**,
* email: **String**,
* roles: [**String**]
* }, { autoCreate: false, autoIndex: false });
* **const** **User** = mongoose.**model**('User', userSchema);
* **const** **RedactedUser** = mongoose.**model**('RedactedUser', userSchema);
* *// First, create the User model's underlying collection...*
* **await** **User**.**createCollection**();
* *// Then create the `RedactedUser` model's underlying collection*
* *// as a View.*
* **await** **RedactedUser**.**createCollection**({
* viewOn: 'users', *// Set `viewOn` to the collection name, \*\*not\*\* model name.*
* pipeline: [
* {
* $set: {
* name: { $concat: [{ $substr: ['$name', 0, 3] }, '...'] },
* email: { $concat: [{ $substr: ['$email', 0, 3] }, '...'] }
* }
* }
* ]
* });
* **await** **User**.**create**([
* { name: 'John Smith', email: 'john.smith@gmail.com', roles: ['user'] },
* { name: 'Bill James', email: 'bill@acme.co', roles: ['user', 'admin'] }
* ]);
* *// [{ \_id: ..., name: 'Bil...', email: 'bil...', roles: ['user', 'admin'] }]*
* console.**log**(**await** **RedactedUser**.**find**({ roles: 'admin' }));

References:

1. Garuba, O. M. (n.d.). *5 steps to authenticating node.js with JWT*. Codementor. Retrieved March 19, 2023, from https://www.codementor.io/@olatundegaruba/5-steps-to-authenticating-node-js-with-jwt-7ahb5dmyr

2. *Models*. Mongoose v7.0.2: Models. (n.d.). Retrieved March 19, 2023, from https://mongoosejs.com/docs/models.html

1. Garuba, O. M. (n.d.). *5 steps to authenticating node.js with JWT*. Codementor. Retrieved March 19, 2023, from https://www.codementor.io/@olatundegaruba/5-steps-to-authenticating-node-js-with-jwt-7ahb5dmyr [↑](#footnote-ref-1)
2. *Models*. Mongoose v7.0.2: Models. (n.d.). Retrieved March 19, 2023, from https://mongoosejs.com/docs/models.html [↑](#footnote-ref-2)