Day04 Authorization 02

part 1 – Install JWT and create User Collection

1. Copy the folder called **Day04-Deployment-01** from **Day04** paste it and rename it to **Day04-Authorization-02**.
2. Run **mpn install** **FIRST** on the new folder then run the following command to install JWD

**npm install jsonwebtoken**

1. Go into the **models** folder and copy the employees.js file paste and rename to users.js
2. Change the users.js file contents to reflect a user instead of weights:

|  |
| --- |
| **const mongoose = require('mongoose');**  **mongoose.connect('mongodb://localhost:27017/Weights', { useNewUrlParser: true });**  **const wSchema = new mongoose.Schema({**  **empName: String,**  **empPass: String**  **},{**  **collection:'EmployeeUsers'**  **});**  **module.exports = mongoose.model('Users', wSchema);** |

part 2 – Add User Functionality

1. In controllers.js file import the **EmployeeUsers** collection, the same way we imported the Weights collection

|  |
| --- |
| **const Weight = require('../models/employees');**  **const User = require('../models/users');**  **const path = require("path");** |

1. In controllers.js file, copy the function **putnewdoc** and change its name to **putnewuser**. Also change the contents of **putnewuser** to reflect adding users instead of employee weights

|  |
| --- |
| **exports.putnewuser = function(req,res){**  **let empName = req.body.empName;**  **let empPass = req.body.empPass;**  **const user = new User();**  **user.empName = empName;**  **user.empPass = empPass;**  **user.save({}, function(err) {**  **if (err)**  **res.end(err);**  **res.end(`Created new user ${empName}`);**  **});**  **};** |

1. Also import the **jsonwebtoken** package

|  |
| --- |
| **const jwt = require('jsonwebtoken');**  **const Weight = require('../models/employees');**  **const User = require('../models/users');** |

1. In controllers.js file, copy the **putnewuser** function and rename it to **loginuser**. This function will handle logging in of users. There is no need to logout a user with a JWT solution, the token simply expires. Also remove everything except the first two lines.

|  |
| --- |
| **exports.loginuser = function(req,res){**  **let empName = req.body.empName;**  **let empPass = req.body.empPass;**  **};** |

1. Now implement the **find()** function to find the user seeking access (or a token in this case)

|  |
| --- |
| **exports.loginuser = function(req,res){**  **let empName = req.body.empName;**  **let empPass = req.body.empPass;**  **User.find({empName:empName}, function(err, results){**  **if (err)**  **res.end(err);**  **});**  **};** |

1. If we find the user, then we proceed otherwise we send back to the client that the login failed. Notice I am just checking the passwords here

|  |
| --- |
| **exports.loginuser = function(req,res){**  **let empName = req.body.empName;**  **let empPass = req.body.empPass;**  **User.find({empName:empName}, function(err, results){**  **if (err)**  **res.end(err);**  **if(results[0].empPass == empPass){**  **//user exists so now use jsonwebtoken**  **} else {**  **res.end("Login failed");**  **}**  **});**  **};** |

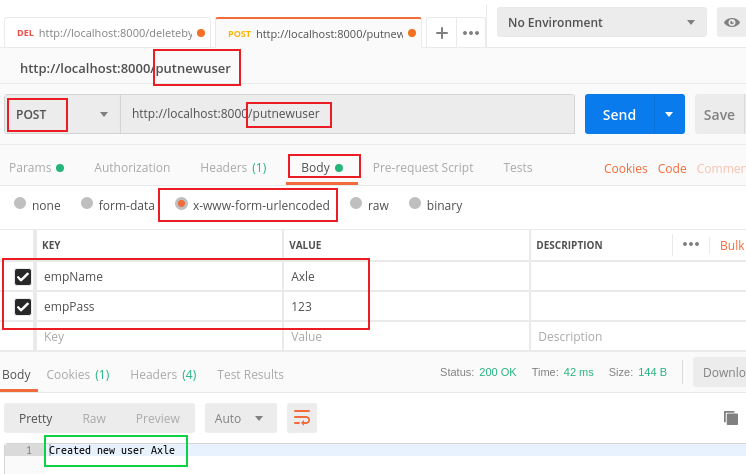
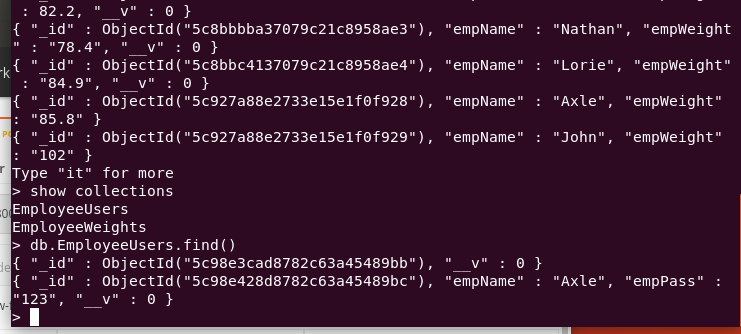
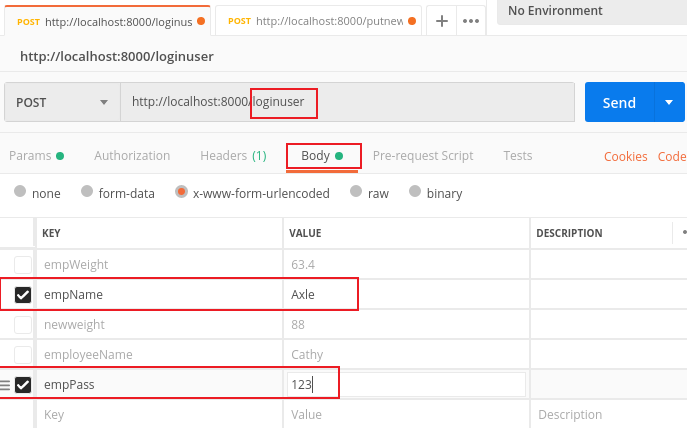
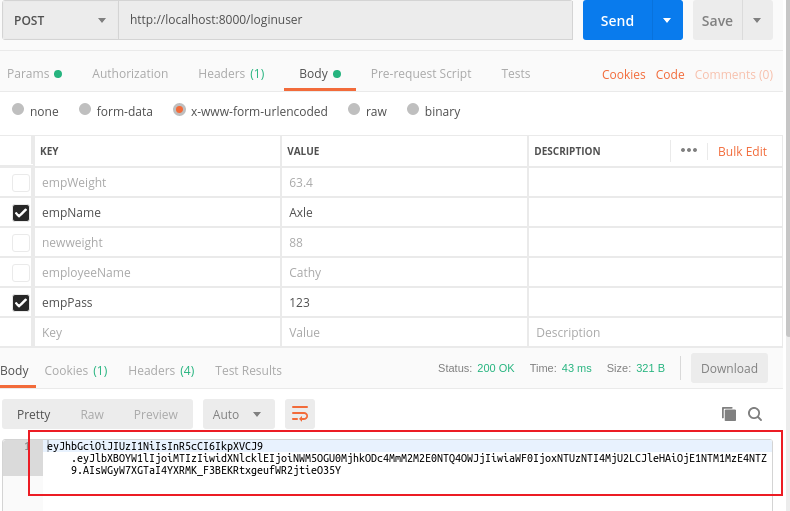
1. If the passwords match then we have a valid user, we can use the **jwt** object to generate a token and send it back to the requesting client

|  |
| --- |
| **User.find({empName:empName}, function(err, results){**  **if (err)**  **res.end(err);**  **if(results[0].empPass == empPass){**  **//user exists so now use jsonwebtoken**  **jwt.sign({**  **empName:results[0].empPass,**  **userID:results[0].\_id**  **},**  **"mysecret",**  **{expiresIn : "1h"},**  **function(err, token){**  **if(err) throw err;**  **res.end(token);**  **}**  **)**  **} else {**  **res.end("Login failed");**  **}**  **});** |

The sign() method takes several parameters. Once find() from mongo returns, it will contain the name and password, so we extract that using the array syntax and passing that as a payload to the sign() method of jwt. We also pass a some text which will act as our key, then an expiry date and finally a function that asynchronously returns the token to the client.

1. In routes.js file, add two routes to handle user signup and login, the controller functions already exist

|  |
| --- |
| **app.route('/updatedoc').put(controller. updatedoc);**  **app.route('/getbyformname').post(controller.getbyformname);**  **app.route('/putnewuser').post(controller.putnewuser);**  **app.route('/loginuser').post(controller.loginuser);**  **//** |

1. Test adding a new user using the REST client  
   
2. You may also verify by logging into the MongoDB CLI and checking for the EmployeeUsers collection  
     
   
3. Now that we have a user, lets sign in that user to see if a token can be generated. The first step in this process is to use the REST client with the empName and empPass fields filled out, along with the url and restful method:  
   
4. The second step is to hit send to generate the token:  
     
   

part 2 – Adding Authorization Middleware

1. We now need to add middleware to the call stack in order to verify the token being passed by a user. In the controllers folder add a new js file called auth.js, then start with the following boilerplate code:

|  |
| --- |
| **const jwt = require("jsonwebtoken");**  **module.exports = function (req, res, next) {**  **}** |

1. Although tokens can be sent in several ways, it is conventional to send them via the headers file of a request. Lets create a new variable to hold that token value from the authorization headers.

|  |
| --- |
| **const jwt = require("jsonwebtoken");**  **module.exports = function (req, res, next) {**  **const rawToken = req.headers.authorization;**  **}** |

1. We can now use the jwt object to verify the token we just got from the headers section of the request

|  |
| --- |
| **const jwt = require("jsonwebtoken");**  **module.exports = function (req, res, next) {**  **const rawToken = req.headers.authorization;**  **const decToken = jwt.verify(rawToken, 'mysecret');**  **}** |

Notice that we also have to pass the key to the verify function as the second parameter

1. We can now store the decToken in a response object ready for sending back to the client. At the same time call the **next()** function so that the request-response cycle can continue on its journey

|  |
| --- |
| **const jwt = require("jsonwebtoken");**  **module.exports = function (req, res, next) {**  **const rawToken = req.headers.authorization;**  **const decToken = jwt.verify(rawToken, 'mysecret');**  **}** |

1. This code as it is will not work, remember the authorization header contains some extra information by convention, it has the word Bearer then a space then the actual toke, we need to extract only the token.

|  |
| --- |
| **const jwt = require("jsonwebtoken");**  **module.exports = function (req, res, next) {**  **const rawToken = req.headers.authorization.split(" ")[1];**  **const decToken = jwt.verify(rawToken, 'mysecret');**  **}**  **}** |

1. Since this is middleware, we have access to the response and request objects, we could pass back to controller via the request object the token we just received, although it is not necessary in this case. Also call the next() function in the call stack.

|  |
| --- |
| **const jwt = require("jsonwebtoken");**  **module.exports = function (req, res, next) {**  **const rawToken = req.headers.authorization.split(" ")[1];**  **const decToken = jwt.verify(rawToken, 'mysecret');**  **req.userInfo = decToken;**  **next();**  **}**  **}** |

1. The JWT does not have native error handlers, so wrap up the code in try catch block for safety

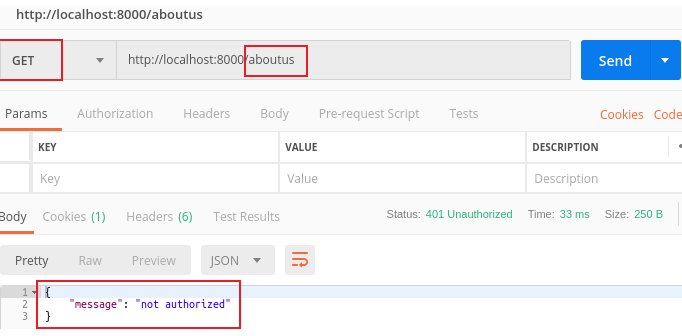
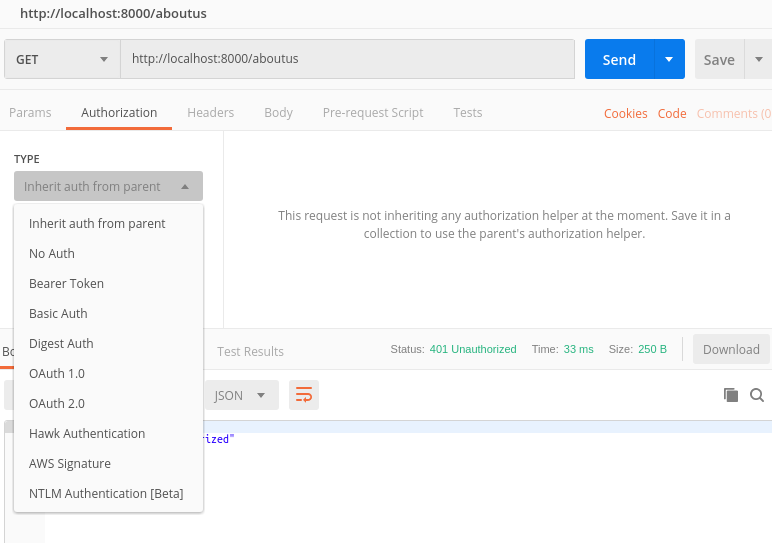
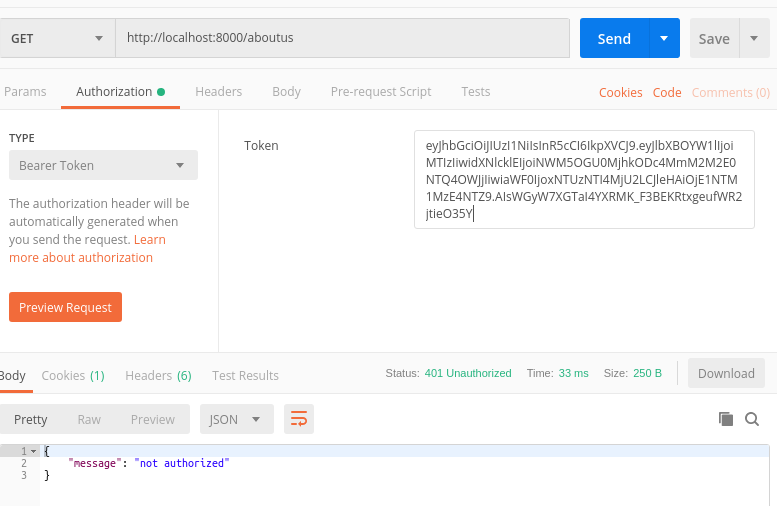
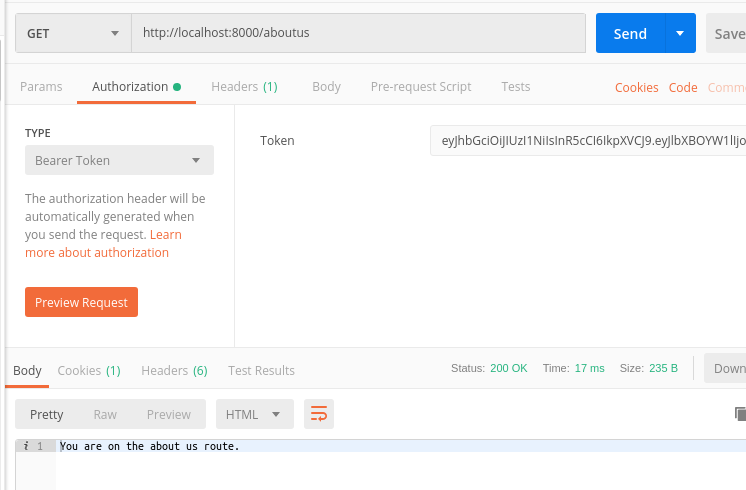
|  |
| --- |
| **const jwt = require("jsonwebtoken");**  **module.exports = function (req, res, next) {**  **try{**  **const rawToken = req.headers.authorization.split(" ")[1];**  **const decToken = jwt.verify(rawToken, 'mysecret');**  **req.userInfo = decToken;**  **next();**  **}catch(error){**  **return res.status(401).json({message:"not authorized"});**  **}**  **}** |

1. All that’s left now is to protect a route, first import the auth.js file we just created into the routes.js file

|  |
| --- |
| **module.exports = function(app){**  **const express= require('express');**  **app.use(express.static(\_\_dirname + '/../HTML'));**  **let controller = require('../controllers/controller');**  **let authUser = require('../controllers/auth');**  **app.route('/').get(controller.getdefault);**  **app.route('/aboutus').get(controller.aboutus);**  **app.route('/employees/:employeeName').get(controller.employees);** |

1. We will experiment with the aboutus route, in terms of protecting this route. Simply insert the authUser variable before the controller part

|  |
| --- |
| **app.use(express.static(\_\_dirname + '/../HTML'));**  **let controller = require('../controllers/controller');**  **let authUser = require('../controllers/auth');**  **app.route('/').get(controller.getdefault);**  **app.route('/aboutus').get(authUser, controller.aboutus);**  **app.route('/employees/:employeeName').get(controller.employees);**  **app.route('/getallrecords').get(controller.getallrecords);** |

1. Now we can test first without the token:  
   
2. Now lets make the same request by passing in the token we generated on a previous tab  
     
   
3. Choose Bearer Token and you will get a small box to enter the token from a previous tab.   
     
   11. Now you can hit the send button  
   
4. You may try to manually change the token, for example remove the first “e” and hit send, the request will be denied