**Authentication - Express @ 18 Jun 2019**

Express-ACL (access control list)

ACL can set up a list of user roles and specify a list of actions that each role is able to perform on a particular resource

npm i express-acl

Set up the configuration of who can access what. Express-acl will look for a acl.yaml file in the root folder of your application, wherever the package.json is located. (Not in the routes folder)

Create in root of application > acl.yml

IN acl.yml, you start off with three items

* yml is dependent on **indentation** to know what’s child and what’s parent relationships
* We can use json to specify the roles as an alternative

Log out is not require because once you are logged in you are no longer a guest. You don’t need a non-logged in user to have access to ‘log out’.

---

- group: admin

permissions:

- resource: "\*"

methods: "\*"

action: allow

- group: user

permissions:

- resource: dashboard/\*

methods: "\*"

action: deny

- resource: "\*"

methods: "\*"

action: allow

- group: guest

permissions:

- resource: "/"

methods:

- GET

action: allow

- resource: users/login

methods:

- POST

action: allow

- resource: users/register

methods:

- POST

action: allow

If we need to change the permissions at any time in the future (add new routes, new user groups), we only need to change acl.yml and don’t have to modify any other code. That makes configuration a lot more maintainable and easier.

Go to app.js, add to the top

const acl = require('express-acl')

Place before your routes in app.js, add the configuration for express-acl . move this below the sessions configuration in app.js

acl.config({

filename: 'acl.yml',

defaultRole: 'guest'

})

app.use(acl.authorize)

add routes to app.js

app.use('/dashboard', secure, adminRouter) // secure = you have to be logged in to access this route (this exclude anyone who is not logged in)

add adminRouter route in app.js

var adminRouter = require('./routes/dashboard')

in app.js add

const secure = require('./middleware/secure')

in routes > index.js and rename to dashboard.js

add to dashboard.js

var express = require('express')

var router = express.Router()

router.get('/', function (req, res, next) {

res.send('<h1>Admin Dashboard</h1>')})

module.exports = router

Test on Postman

* Test the acl rules on different methods on different urls
* Localhost:3000/dashboard as a GET message should return a json error object to say access is denied and with a 403 status code for forbidden access

Need some mechanism to give a user a role. Only allow user have a single role at any one time. In more sophisticated systems, a user can have multiple roles and inherit permissions from all of those roles.

In user model user.js

const mongoose = require('mongoose')

const passportLocalMongoose = require('passport-local-mongoose')

const UserSchema = new mongoose.Schema({

role: string

})

UserSchema.plugin(passportLocalMongoose, {

usernameField: 'email'

})

const UserModel = mongoose.model('User', UserSchema)

module.exports = {

UserSchema,

UserModel

}

Database-agnostic. Express-acl is when a request comes in, in order to know the role of the current user, it will look for an attribute called ‘role’ that is in session.

It is up to us to set that value when the user logs in. When a user logs in , we want to get their role out of the model and set a session variable called ‘role’ which has that same value.

IN users.js, modify the UserModel register callback method for authenticating and also when logging in (POST)

( *Refer back to the video for details* )

Passport.authenticate(‘local’)(req, res, () => {

req.session.role = req.user.role = ‘guest’

res.json(req.json)

})

router.post(‘login’, passport.authetnciate(‘local’), (req, res) => {

req.session.role = req.user.role = ‘guest’

res.json(req.user)

})

There are situations where existing users haven’t been set a role. So we need to add a OR operator and include ‘guest’ for unauthenticated user.

Test this change on POSTMAN and should still get a deny/forbidden status.

To make the user an admin, we have to do it in the terminal because we don’t yet have an interface in the application.

* mongo (goes into the mongo cli)
* Show dbs
* Use passport-demo
* Show collections
* db.users.find() //reveals users
* db.users.update({ email: ‘foo@bar.com’ }, { $set: { role: ‘admin’ } }) // changes the admin role of the user based on the email criteria. This is the same user that you just created earlier in the form.
* db.users.find() // reveals all the users with a role admin updated

Try this POSTMAN again, log in as the user with the email address assigned as admin (using POST in localhost:3000/users/login). Access the dashboard (localhost:3000/dashboard), and you should

On Postman, go to localhost:3000/users/register

* create a new user (non admin)

Go back into mongo terminal CLI, change the user you just created a role of ‘user’

* db.users.update({ email: ‘user@bar.com’ }, { $set: { role: ‘user’ }})

And test to go into dashboard, the user should be denied (as per acl.yml)

\* With a newly registered user could be limited in what they could do until the admin approves the user with a role designation. Until they are approved, they have limited permissions. Or an automated email system which will get users to click on an encrypted link. Initially you make them a guest, until you have another route that receives the encrypted value and check it against the user, if they match then make them a full ‘user’ role account.

**Customise the forbidden access message**

When we are denied, we receive the JSON string of denial of access. We can customise the message.

Go into app.js, amend the acl.config part with the following

Note that the new *denyCallback* function can actually res.render a view instead of just spitting out a text message (res.status).

acl.config({

filename: 'acl.yml',

defaultRole: 'guest',

denyCallback: (res) => {

// instead of acl call internal function, it wil call the denyCallback and pass the response object

// this can be a res.render to render a view for forbidden message

res.status(403).json({

status: 'Verboten!',

success: false,

message: 'You are not authorised to do that!'

// these are the same three keys from the default forbidden json string

})

}

})