# NODE ACL - Access Control Lists for Node

This module provides a minimalistic ACL implementation inspired by Zend\_ACL.

When you develop a web site or application you will soon notice that sessions are not enough to protect all the available resources. Avoiding that malicious users access other users content proves a much more complicated task than anticipated. ACL can solve this problem in a flexible and elegant way.

Create roles and assign roles to users. Sometimes it may even be useful to create one role per user, to get the finest granularity possible, while in other situations you will give the asteriskpermission for admin kind of functionality.

A Redis, MongoDB and In-Memory based backends are provided built-in in the module. There are other third party backends such as [knex](https://github.com/christophertrudel/node_acl_knex) based, [firebase](https://github.com/tonila/node_acl_firebase) and [elasticsearch](https://github.com/adnanesaghir/acl-elasticsearch-backend). There is also an alternative memory backend that supports [regexps](https://github.com/futurechan/node_acl-mem-regexp).

Follow [manast](http://twitter.com/manast) for news and updates regarding this library.

## Status

[BuildStatus](http://travis-ci.org/OptimalBits/node_acl)

## Features

* Users
* Roles
* Hierarchies
* Resources
* Express middleware for protecting resources.
* Robust implementation with good unit test coverage.

## Installation

Using npm:

npm install acl

## Documentation

* [addUserRoles](https://www.npmjs.com/package/acl#addUserRoles)
* [removeUserRoles](https://www.npmjs.com/package/acl#removeUserRoles)
* [userRoles](https://www.npmjs.com/package/acl#userRoles)
* [roleUsers](https://www.npmjs.com/package/acl#roleUsers)
* [hasRole](https://www.npmjs.com/package/acl#hasRole)
* [addRoleParents](https://www.npmjs.com/package/acl#addRoleParents)
* [removeRoleParents](https://www.npmjs.com/package/acl#removeRoleParents)
* [removeRole](https://www.npmjs.com/package/acl#removeRole)
* [removeResource](https://www.npmjs.com/package/acl#removeResource)
* [allow](https://www.npmjs.com/package/acl#allow)
* [removeAllow](https://www.npmjs.com/package/acl#removeAllow)
* [allowedPermissions](https://www.npmjs.com/package/acl#allowedPermissions)
* [isAllowed](https://www.npmjs.com/package/acl#isAllowed)
* [areAnyRolesAllowed](https://www.npmjs.com/package/acl#areAnyRolesAllowed)
* [whatResources](https://www.npmjs.com/package/acl#whatResources)
* [middleware](https://www.npmjs.com/package/acl#middleware)
* [backend](https://www.npmjs.com/package/acl#backend)

## Examples

Create your acl module by requiring it and instantiating it with a valid backend instance:

var acl **=** require('acl');

*// Using redis backend*

acl **=** **new** acl(**new** acl.redisBackend(redisClient, prefix));

*// Or Using the memory backend*

acl **=** **new** acl(**new** acl.memoryBackend());

*// Or Using the mongodb backend*

acl **=** **new** acl(**new** acl.mongodbBackend(dbInstance, prefix));

All the following functions return a promise or optionally take a callback with an err parameter as last parameter. We omit them in the examples for simplicity.

Create roles implicitly by giving them permissions:

*// guest is allowed to view blogs*

acl.allow('guest', 'blogs', 'view')

*// allow function accepts arrays as any parameter*

acl.allow('member', 'blogs', ['edit', 'view', 'delete'])

Users are likewise created implicitly by assigning them roles:

acl.addUserRoles('joed', 'guest')

Hierarchies of roles can be created by assigning parents to roles:

acl.addRoleParents('baz', ['foo', 'bar'])

Note that the order in which you call all the functions is irrelevant (you can add parents first and assign permissions to roles later)

acl.allow('foo', ['blogs', 'forums', 'news'], ['view', 'delete'])

Use the wildcard to give all permissions:

acl.allow('admin', ['blogs', 'forums'], '\*')

Sometimes is necessary to set permissions on many different roles and resources. This would lead to unnecessary nested callbacks for handling errors. Instead use the following:

acl.allow([

    {

        roles**:**['guest', 'member'],

        allows**:**[

            {resources**:**'blogs', permissions**:**'get'},

            {resources**:**['forums', 'news'], permissions**:**['get', 'put', 'delete']}

        ]

    },

    {

        roles**:**['gold', 'silver'],

        allows**:**[

            {resources**:**'cash', permissions**:**['sell', 'exchange']},

            {resources**:**['account', 'deposit'], permissions**:**['put', 'delete']}

        ]

    }

])

You can check if a user has permissions to access a given resource with isAllowed:

acl.isAllowed('joed', 'blogs', 'view', function(err, res){

**if**(res){

        console.log("User joed is allowed to view blogs")

    }

})

Of course arrays are also accepted in this function:

acl.isAllowed('jsmith', 'blogs', ['edit', 'view', 'delete'])

Note that all permissions must be fulfilled in order to get true.

Sometimes is necessary to know what permissions a given user has over certain resources:

acl.allowedPermissions('james', ['blogs', 'forums'], function(err, permissions){

    console.log(permissions)

})

It will return an array of resource:[permissions] like this:

[{'blogs' **:** ['get', 'delete']},

 {'forums'**:**['get', 'put']}]

Finally, we provide a middleware for Express for easy protection of resources.

acl.middleware()

We can protect a resource like this:

app.put('/blogs/:id', acl.middleware(), function(req, res, next){…}

The middleware will protect the resource named by req.url, pick the user from req.session.userId and check the permission for req.method, so the above would be equivalent to something like this:

acl.isAllowed(req.session.userId, '/blogs/12345', 'put')

The middleware accepts 3 optional arguments, that are useful in some situations. For example, sometimes we cannot consider the whole url as the resource:

app.put('/blogs/:id/comments/:commentId', acl.middleware(3), function(req, res, next){…}

In this case the resource will be just the three first components of the url (without the ending slash).

It is also possible to add a custom userId or check for other permissions than the method:

app.put('/blogs/:id/comments/:commentId', acl.middleware(3, 'joed', 'post'), functi

# [**node\_acl\_example.js**](https://gist.github.com/danwit/11307969#file-node_acl_example-js)

|  |
| --- |
| /\*\* |
|  | \* Simple node\_acl example with mongoDB and expressjs |
|  | \* |
|  | \* Usage: |
|  | \* 1. Start this as server |
|  | \* 2. Play with the resoures |
|  | \* |
|  | \* Show all permissions (as JSON) |
|  | \* http://localhost:3500/info |
|  | \* |
|  | \* Only visible for users and higher |
|  | \* http://localhost:3500/secret |
|  | \* |
|  | \* Only visible for admins |
|  | \* http://localhost:3500/topsecret |
|  | \* |
|  | \* Manage roles |
|  | \* user is 'bob' and role is either 'guest', 'user' or 'admin' |
|  | \* http://localhost:3500/allow/:user/:role |
|  | \* http://localhost:3500/disallow/:user/:role |
|  | \* |
|  | \* Don't forget to disallow a role, if you want to revoke its |
|  | \* permissions. |
|  | \*/ |
|  |  |
|  | var mongodb = require( 'mongodb' ), |
|  | express = require( 'express' ), |
|  | node\_acl = require( 'acl' ), |
|  | port = 3500, |
|  | app = express(), |
|  | // The actual acl will reside here |
|  | acl; |
|  |  |
|  | // Error handling ( most notably 'Insufficient permissions' ) |
|  | app.use( app.router ); |
|  | app.use( function( error, request, response, next ) { |
|  | if( !error ) return next(); |
|  | response.send( error.msg, error.errorCode ); |
|  | }); |
|  |  |
|  | // Connecting to our mongo database |
|  | mongodb.connect( 'mongodb://127.0.0.1:27017/acl\_example', \_mongo\_connected ); |
|  |  |
|  | function \_mongo\_connected( error, db ) { |
|  |  |
|  | var mongoBackend = new node\_acl.mongodbBackend( db /\*, {String} prefix \*/ ); |
|  |  |
|  | // Create a new access control list by providing the mongo backend |
|  | // Also inject a simple logger to provide meaningful output |
|  | acl = new node\_acl( mongoBackend, logger() ); |
|  |  |
|  | // Defining roles and routes |
|  | set\_roles(); |
|  | set\_routes(); |
|  | } |
|  |  |
|  | // This creates a set of roles which have permissions on |
|  | // different resources. |
|  | function set\_roles() { |
|  |  |
|  | // Define roles, resources and permissions |
|  | acl.allow([ |
|  | { |
|  | roles: 'admin', |
|  | allows: [ |
|  | { resources: '/secret', permissions: 'create' }, |
|  | { resources: '/topsecret', permissions: '\*' } |
|  | ] |
|  | }, { |
|  | roles: 'user', |
|  | allows: [ |
|  | { resources: '/secret', permissions: 'get' } |
|  | ] |
|  | }, { |
|  | roles: 'guest', |
|  | allows: [] |
|  | } |
|  | ]); |
|  |  |
|  | // Inherit roles |
|  | // Every user is allowed to do what guests do |
|  | // Every admin is allowed to do what users do |
|  | acl.addRoleParents( 'user', 'guest' ); |
|  | acl.addRoleParents( 'admin', 'user' ); |
|  | } |
|  |  |
|  | // Defining routes ( resources ) |
|  | function set\_routes() { |
|  |  |
|  | // Simple overview of granted permissions |
|  | app.get( '/info', |
|  | function( request, response, next ) { |
|  | acl.allowedPermissions( get\_user\_id(), [ '/info', '/secret', '/topsecret' ], function( error, permissions ){ |
|  | response.json( permissions ); |
|  | }); |
|  | } |
|  | ); |
|  |  |
|  | // Only for users and higher |
|  | app.get( '/secret', acl.middleware( 1, get\_user\_id ), |
|  | function( request, response, next ) { |
|  | response.send( 'Welcome Sir!' ); |
|  | } |
|  | ); |
|  |  |
|  | // Only for admins |
|  | app.get( '/topsecret', acl.middleware( 1, get\_user\_id ), |
|  | function( request, response, next ) { |
|  | response.send( 'Hi Admin!' ); |
|  | } |
|  | ); |
|  |  |
|  | // Setting a new role |
|  | app.get( '/allow/:user/:role', function( request, response, next ) { |
|  | acl.addUserRoles( request.params.user, request.params.role ); |
|  | response.send( request.params.user + ' is a ' + request.params.role ); |
|  | }); |
|  |  |
|  | // Unsetting a role |
|  | app.get( '/disallow/:user/:role', function( request, response, next ) { |
|  | acl.removeUserRoles( request.params.user, request.params.role ); |
|  | response.send( request.params.user + ' is not a ' + request.params.role + ' anymore.' ); |
|  | }); |
|  | } |
|  |  |
|  | // Provide logic for getting the logged-in user |
|  | // This is a job for your authentication layer |
|  | function get\_user\_id( request, response ) { |
|  | return 'bob'; |
|  | } |
|  |  |
|  | // Generic debug logger for node\_acl |
|  | function logger() { |
|  | return { |
|  | debug: function( msg ) { |
|  | console.log( '-DEBUG-', msg ); |
|  | } |
|  | }; |
|  | } |
|  |  |
|  | // Starting the server |
|  | app.listen( port, function() { |
|  | console.log( 'ACL example listening on port ' + port ); |
|  | }); |

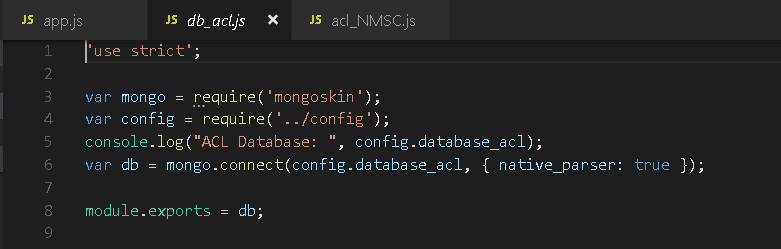
# RealTime example –

var node\_acl = require('acl');

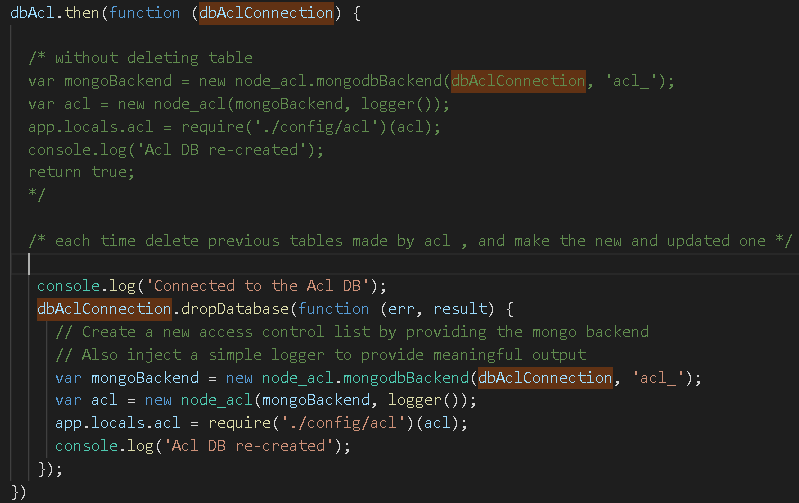
var dbAcl = require('./helpers/db\_acl');

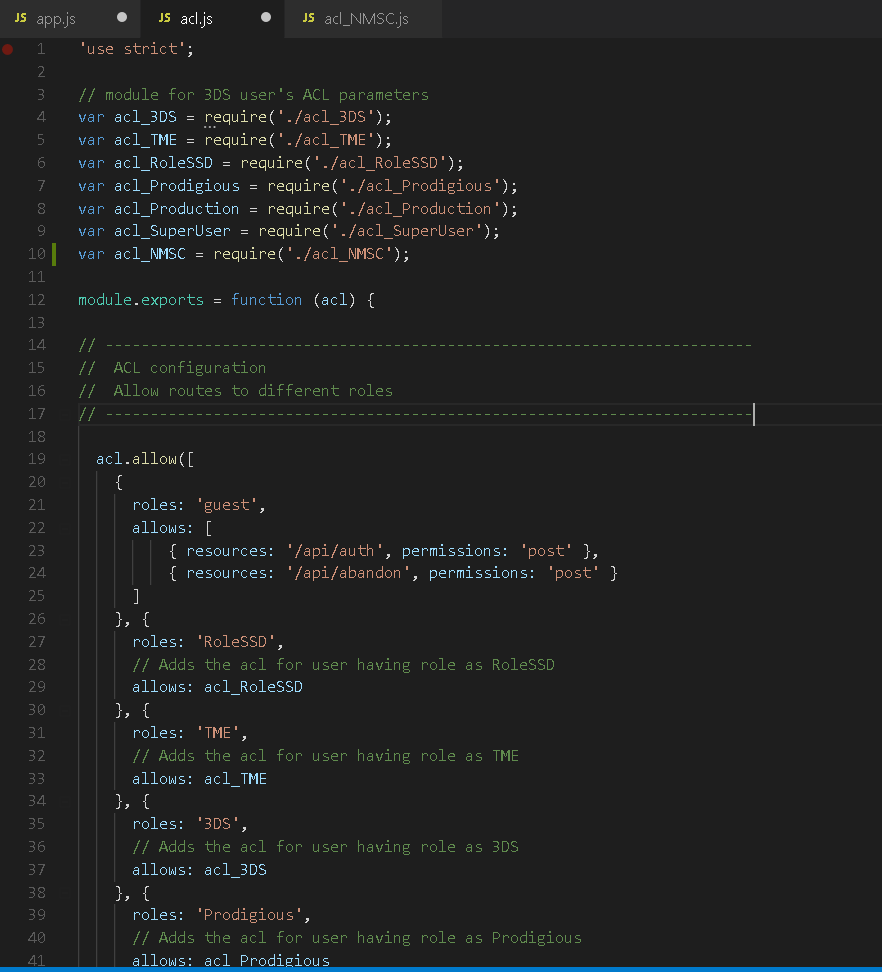
--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

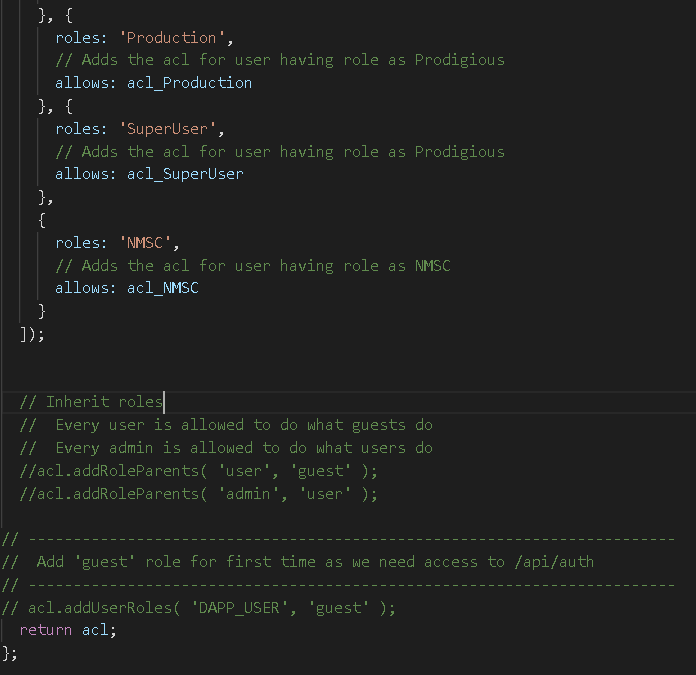
dbAcl is –



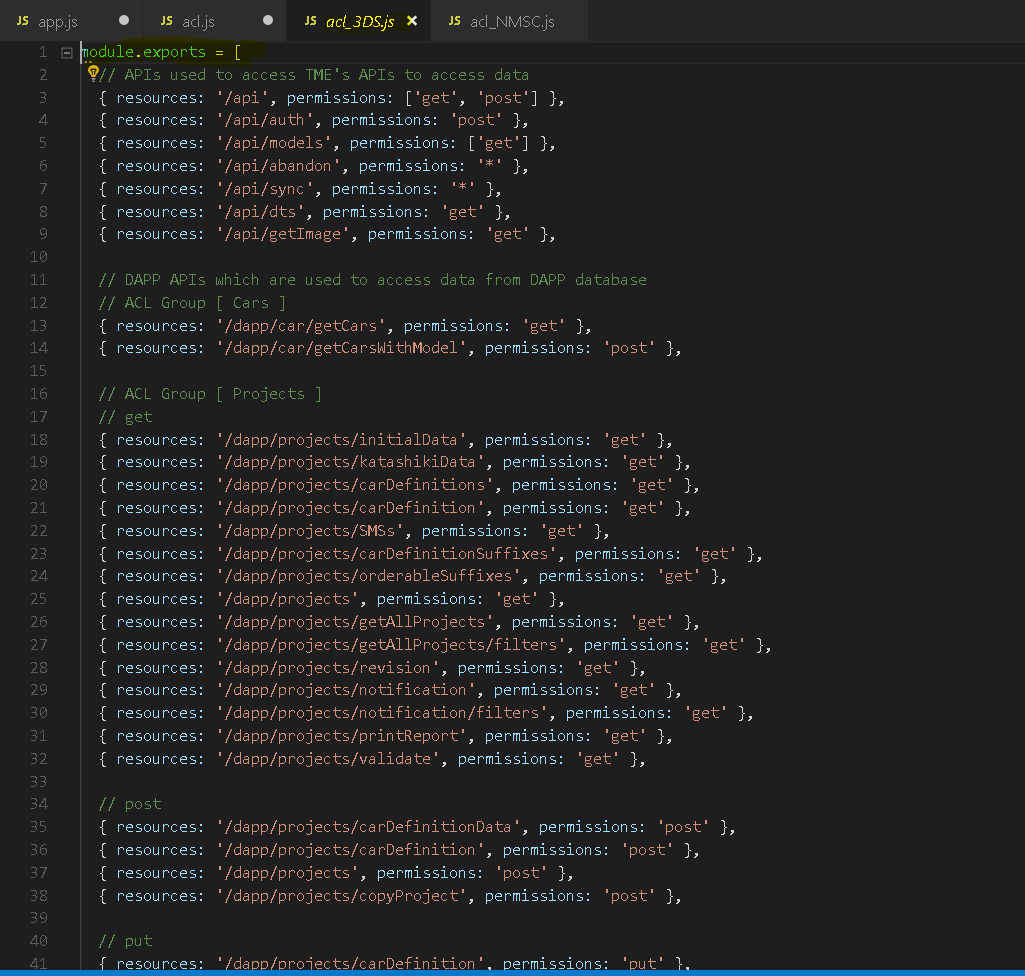
----\*\*\*\*\*\*\*\*\*\*\*\*



Config/acl -



Take any file for example **acl\_3DS**

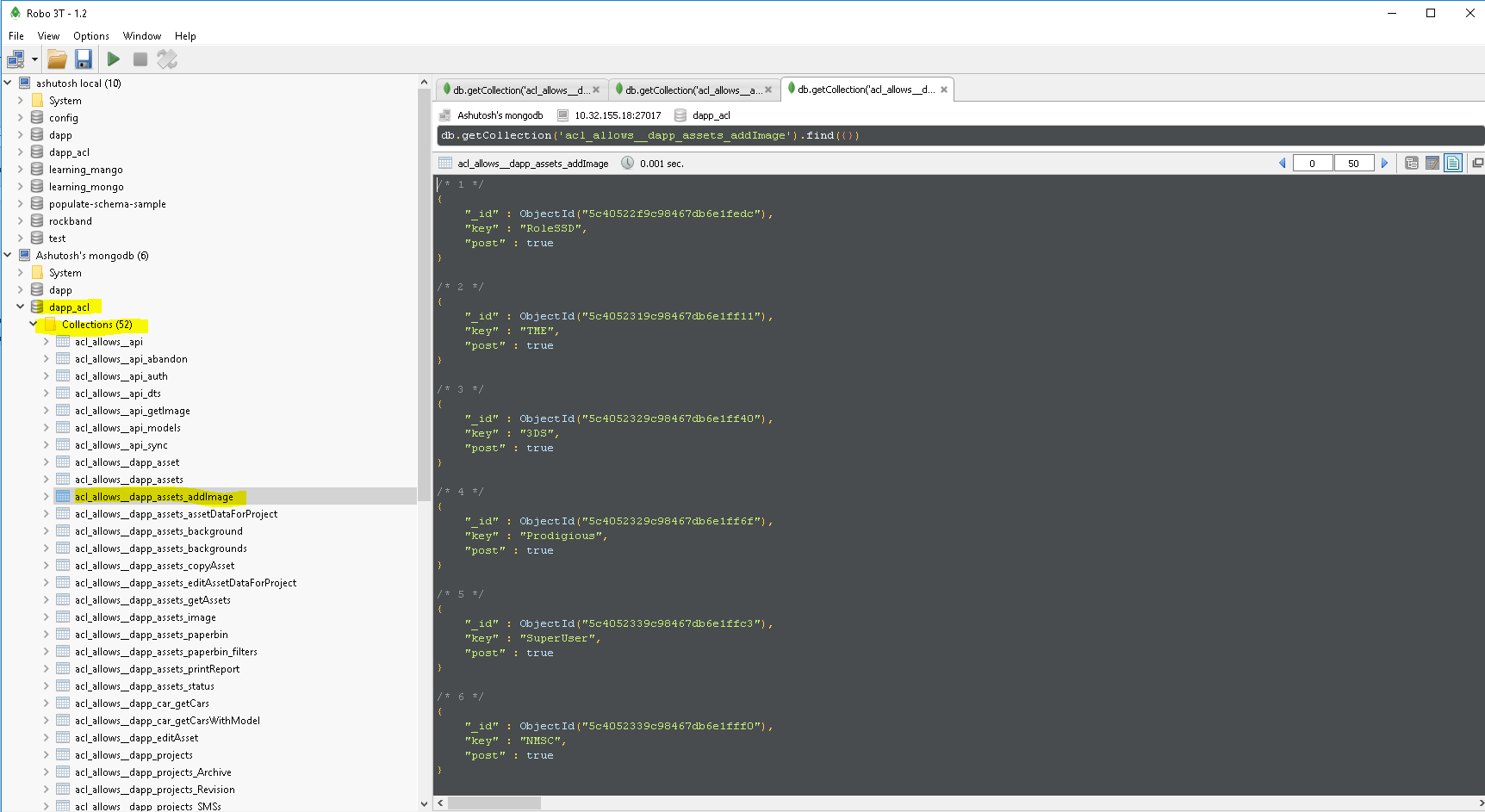
**For giving access to many resources , we have made a js file to provide the allows array**

**Note –**

**acl.allow will create different collections according to resource path provided inside provided Database.**

**For example –**

**For resource “**resources: '/dapp/assets/image', permissions: 'get' },**”**



**So acl create as many as collection in provided database for particular resource and add roles as record in it.**

**After that you need to add user roles for logged in user and according to role of logged in user, acl will give access to any resource.**

**acl.addUserRoles( 'DAPP\_USER', 'NMSC' );**