2018-06-23T03:38:13.130Z | Mahesh Bhanushali

Building a Restful CRUD API with Node.js, JWT, Bcrypt, Express and MongoDB

Introduction

In this tutorial we will be developing simple REST API for movies collection with their released date. We gonna implement simple CRUD(Create, Read, Update and Delete) operations on movie collection data in our journey.

Our application will have public API as well as protected API when we say public that mean it can be accessed without any authentication whereas protected we need to authenticate our self to access them. For authentication, we will be using JWT(Json Web Token). Movie collection application will have the following workflow

- User register through the registration form. (Name, Email, Password)
- The user is authenticated by providing email and password
- On successful authentication, JsonWebToken(JWT) is returned in response to access protected routes.
- In subsequent access to protected route, the user sends JWT token in the header for authentication instead of email and password.

Prerequisite

Before we start with coding following list needs to be installed. I am working on windows environment.

- nodejs
- npm
- mongodb

> For windows environment MongoDB community version is recommended for quick start

I assume above list has been installed successfully on your development environment

Implementation

Open terminal and navigate to directory where you want your application to stay and follow following steps

mkdir node-rest-api-jwt
cd node-rest-api-jwt
npm init

After **npm init** you will be asked few questions one by one, keep default options as it is and keep hitting enter button

package name: (node-rest-api-jwt)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\NodejsTut\node-rest-api-jwt\package.json:
{
“name”: “node-rest-api-jwt”,
“version”: “1.0.0”,
“description”: “”,
“main”: “index.js”,
“scripts”: {
“test”: “echo \”Error: no test specified\” && exit 1"
},
“author”: “”,
“license”: “ISC”
}Is this OK? (yes)

Now install following dependencies

npm install bcrypt body-parser express jsonwebtoken mongoose morgan --savenpm install nodemon -g

- body-parser :- Parse incoming request bodies in a middleware before your handlers, available under the `req.body`property
- express :- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
- jsonwebtoken :- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

> You can read more about JWT

- mongoose :- Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment.
- morgan :- HTTP request logger middleware for node.js
- nodemon :- nodemon will watch the files in the directory in which nodemon was started, and if any files change, nodemon will automatically restart your node application.

After installing dependencies your package.json file will have the list of all installed dependencies

"dependencies": {
"bcrypt": "2.0.1",
"body-parser": "1.18.2",
"express": "4.16.3",
"jsonwebtoken": "8.2.1",
"mongoose": "5.0.13",
"morgan": "1.9.0"
},

Now we are good to start with development. Move to root folder of the application i.e node-rest-api-jwt and create following directory

cd node-rest-api-jwt
mkdir app
cd app
mkdir api
cd api
mkdir controllers

We will follow MVC pattern so our application controllers and models files will be stored in app/api/controllers, app/api/models respective directories.

create folder config inside application root directory.

mkdir config

We will store our database configuration file in config folder. i.e node-rest-api-jwt/config

Next we will create route folder inside our application root directory where all of our application related routes files will be created.

mkdir routes

Lets create node server, create a file server.js inside directory node-rest-api-jwt

```
const express = require('express');
const logger = require('morgan');
const bodyParser = require('body-parser');
const app = express();app.use(logger('dev'));
app.use(bodyParser.urlencoded({extended: false}));app.get('/', function(req, res){
res.json({"tutorial" : "Build REST API with node.js"});
});app.listen(3000, function(){ console.log('Node server listening on port 3000');});
```

Now open terminal and from application root folder fire command

nodemon server.js

You will see in terminal

```
[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
Node server listening on port 3000
```

Now open browser and type localhost:3000/

{"tutorial":"Build REST API with node.js"}

```
http://localhost:300
```

Now lets create model file for user in node-rest-api-jwt/app/api/models/users.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const saltRounds = 10;//Define a schema
const Schema = mongoose.Schema;const UserSchema = new Schema({
name: {
type: String,
trim: true,
required: true,
},
email: {
type: String,
trim: true,
required: true
},
password: {
type: String,
trim: true,
required: true
}
});// hash user password before saving into database
UserSchema.pre('save', function(next){
this.password = bcrypt.hashSync(this.password, saltRounds);
next();
});module.exports = mongoose.model('User', UserSchema);
```

In model file we include mongoose for model schema, bcrypt to hash our password and also defined salt round which will be used for hashing plain text password. Mongoose provide middleware(pre/post hooks) which we can use to manipulate our data before/after inserting into database. We have used **pre hook save method** to hash our password before saving into database. In **pre hooks** callback function **this** keyword refer to UserSchema object and this.password is password passed from registration form which we will be creating soon. You can read more about mongoose and its function at https://github.com/Automattic/mongoose

Now create controller for user in node-rest-api-jwt/app/api/controllers/users.js

```
const userModel = require('../models/users');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');module.exports = {
create: function(req, res, next) {

userModel.create({ name: req.body.name, email: req.body.email, password:
req.body.password }, function (err, result) {
if (err)
next(err);
else
res.json({status: "success", message: "User added successfully!!!", data: null});

});
},authenticate: function(req, res, next) {
userModel.findOne({email:req.body.email}, function(err, userInfo){
if (err) {
next(err);
} else {if(bcrypt.compareSync(req.body.password, userInfo.password)) {const token =
jwt.sign({id: userInfo._id}, req.app.get('secretKey'), { expiresIn: '1h'
});res.json({status:"success", message: "user found!!!", data:{user: userInfo,
token:token}});}else{res.json({status:"error", message: "Invalid email/password!!!",
data:null});}
}
});
},}
```

User controller include user model, jsonwebtoken module and bcrypt. And we have defined two methods

- create :- To create new users, this function has model query too create new user

- authenticate :- In this function we search for user in database by email id and compare plain password passed through login form with database hashed password, if password match we generate jwt token by passing user id and secret key, and have set 1hr validity of the token. You can explore more option on jwt at https://jwt.io/introduction/ & https://github.com/auth0/node-jsonwebtoken

Now time to create routes for above users controller methods, node-rest-api-jwt/routes/users.js

```
const express = require('express');
const router = express.Router();
const userController = require('../app/api/controllers/users');router.post('/register',
userController.create);
router.post('/authenticate', userController.authenticate);module.exports = router;
```

In route we included user controller and each post route is mapped with respective controller method.

Create model, controller and route file for protected movie routes as we created for user

node-rest-api-jwt/app/api/controllers/movies.js

```
const movieModel = require('../models/movies');module.exports = {
getById: function(req, res, next) {
console.log(req.body);
movieModel.findById(req.params.movieId, function(err, movieInfo){
if (err) {
next(err);
} else {
res.json({status:"success", message: "Movie found!!!", data:{movies: movieInfo}});
}
});
},getAll: function(req, res, next) {
let moviesList = [];movieModel.find({}, function(err, movies){
if (err){
next(err);
} else{
for (let movie of movies) {
moviesList.push({id: movie._id, name: movie.name, released_on: movie.released_on});
}
res.json({status:"success", message: "Movies list found!!!", data:{movies:
```

```
}});
},updateById: function(req, res, next) {
movieModel.findByIdAndUpdate(req.params.movieId,{name:req.body.name},
function(err, movieInfo){if(err)
next(err);
else {
res.json({status:"success", message: "Movie updated successfully!!!", data:null});
}
});
},deleteById: function(req, res, next) {
movieModel.findByIdAndRemove(req.params.movieId, function(err, movieInfo){
if(err)
next(err);
else {
res.json({status:"success", message: "Movie deleted successfully!!!", data:null});
}
});
},create: function(req, res, next) {
movieModel.create({ name: req.body.name, released_on: req.body.released_on },
function (err, result) {
if (err)
next(err);
else
res.json({status: "success", message: "Movie added successfully!!!", data: null});

});
},}
```

node-rest-api-jwt/app/api/models/movies.js

```
const mongoose = require('mongoose');//Define a schema
const Schema = mongoose.Schema;const MovieSchema = new Schema({
name: {
type: String,
trim: true,
required: true,
},
released_on: {
type: Date,
trim: true,
```

```
}
});module.exports = mongoose.model('Movie', MovieSchema)
```

node-rest-api-jwt/routes/movies.js

```
const express = require('express');
const router = express.Router();
const movieController = require('../app/api/controllers/movies');router.get('/',
movieController.getAll);
router.post('/', movieController.create);
router.get('/:movieId', movieController.getById);
router.put('/:movieId', movieController.updateById);
router.delete('/:movieId', movieController.deleteById);module.exports = router;
```

Open server.js file and update existing code with following

```
const express = require('express');
const logger = require('morgan');
const movies = require('./routes/movies') ;
const users = require('./routes/users');
const bodyParser = require('body-parser');
const mongoose = require('./config/database'); //database configuration
var jwt = require('jsonwebtoken');
const app = express();app.set('secretKey', 'nodeRestApi'); // jwt secret token//
connection to mongodb
mongoose.connection.on('error', console.error.bind(console, 'MongoDB connection
error:'));app.use(logger('dev'));
app.use(bodyParser.urlencoded({extended: false}));app.get('/', function(req, res){
res.json({"tutorial" : "Build REST API with node.js"});
});// public route
app.use('/users', users);// private route
app.use('/movies', validateUser, movies);app.get('/favicon.ico', function(req, res) {
res.sendStatus(204);
});function validateUser(req, res, next) {
jwt.verify(req.headers['x-access-token'], req.app.get('secretKey'), function(err, decoded)
{
if (err) {
res.json({status:"error", message: err.message, data:null});
}else{
// add user id to request
```

```
req.body.userId = decoded.id;
next();
}
});

}// express doesn't consider not found 404 as an error so we need to handle 404
explicitly
// handle 404 error
app.use(function(req, res, next) {
let err = new Error('Not Found');
err.status = 404;
next(err);
});// handle errors
app.use(function(err, req, res, next) {
console.log(err);

if(err.status === 404)
res.status(404).json({message: "Not found"});
else
res.status(500).json({message: "Something looks wrong :( !!!"});});app.listen(3000,
function(){
console.log('Node server listening on port 3000');
});
```

To access movie routes we have defined middleware to validate user

```
// private route
app.use('/movies', validateUser, movies);function validateUser(req, res, next) {
jwt.verify(req.headers['x-access-token'], req.app.get('secretKey'), function(err, decoded)
{
if (err) {
res.json({status:"error", message: err.message, data:null});
}else{
// add user id to request
req.body.userId = decoded.id;
next();
}
});

}
```

Database Connection

Create database connection configuration file in node-rest-api-jwt/config/database.js

```
//Set up mongoose connection
const mongoose = require('mongoose');
const mongoDB = 'mongodb://localhost/node_rest_api';
mongoose.connect(mongoDB);
mongoose.Promise = global.Promise;module.exports = mongoose;
```

and in server.js file we have setup connection to MongoDB

```
// connection to mongodb
mongoose.connection.on('error', console.error.bind(console, 'MongoDB connection error:'));
```

For windows users to start MongoDB server if you have installed community version on your system

Create a data folder in node-rest-api-jwt.

```
mkdir data
```

Now open terminal navigate to mongo folder where you have installed mostly it will be in c:\Program Files\MongoDB\Server\3.4|3.2\bin

```
mongod --dbpath c:\path to node-rest-api-jwt\data
```

This command will start MongoDB server and store all your collection in your application data folder, now open another terminal and hit command

```
nodemon server.js
```

> You can view your MongoDB collection by installing Robo 3T application

One thing I want to add here is that you have noticed we haven't created any collection in MongoDB, so creating collection on fly is handled by mongoose. You can read more on this by exploring mongoose document.

Testing

For testing we will user http client postman

route list

- GET http://localhost:3000
- POST http://localhost:3000/users/register

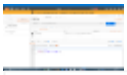
POST http://localhost:3000/users/register

- POST http://localhost:3000/users/authenticate

POST http://localhost:3000/users/authenticate

JWT token received in above request will be used to access protected movie routes. We will pass token in header with key=x-access-token and value=jwt token

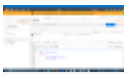
- POST http://localhost:3000/movies



POST http://localhost:3000/movies

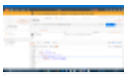
POST http://localhost:3000/movies

- GET http://localhost:3000/movies



http://localhost:3000/movies

- GET http://localhost:3000/movies/5ae63c5027bbe422cce696a3



http://localhost:3000/movies/5ae63c5027bbe422cce696a3

- PUT http://localhost:3000/movies/5ae63c5027bbe422cce696a3

http://localhost:3000/movies/5ae63c5027bbe422cce696a3

http://localhost:3000/movies/5ae63c5027bbe422cce696a3

- DELETE http://localhost:3000/movies/5ae63c5027bbe422cce696a3

Conclusion

In this tutorial we learned to develop REST API with simple CRUD operation, hashing user plain password using bcrypt and JWT token. If have any queries/suggestion please leave a comment and keep following for more such tuts

> You can find complete code on github
