MEAN Stack Application

**What’s Included?**

* REST API – Node.js/Express
* Token Generation & Authentication
* CORS
* Mongoose ODM
* Angular 2/Angular-CLI
* Angular Router, HTTP Module
* Angular2-JWT
* Auth-Guard
* Angular Flash Message Module
* Compile & Deploy

It’s hard to type with long nails and such so we could try going to

Dependencies

"dependencies": {

    "bcryptjs": "^2.4.3",

    "cors": "^2.8.5",

    "express": "^4.17.1",

    "jsonwebtoken": "^8.5.1",

    "mongoose": "^5.9.28",

    "passport": "^0.4.1",

    "passport-jwt": "^4.0.0"

  }

Steps

1. npm init
2. set up dependencies (npm install – save x x x x
3. Set up express and module uses for dependencies

const express = require('express');

const path = require('path');

const cors = require('cors');

const passport = require('passport');

const mongoose = require('mongoose');

1. Create config file

module.exports = {

    database: 'mongodb://localhost:27017/meanauth',

    secret: "yoursecret"

}

1. Set up db

// Connect to MongoDB database

// Config.database is the localhost url

mongoose.connect(config.database, {useNewUrlParser: true, useUnifiedTopology: true});

// Connection Success

mongoose.connection.on('conneection', () => {

    console.log(`Successfully connected to database: ${config.database}`);

});

// Connection Failure

mongoose.connection.on('err', (err) => {

    console.log(`Failed to connect to database: ${config.database}`);

    console.log(`Error: ${err}`)

});

1. Create routes/users.js (user controller)

const express = require('express');

const router = express.Router();

// Register user

router.post('/register', (req, res) => {

    res.send("Register page")

});

1. Set up middleware

// Similar to controller

const usersRoute = require('./routes/users');

// Express cors can also be used

app.use(cors());

// Set static folder. You can also call folder public

// Create a folder

app.use(express.static(path.join(\_\_dirname, 'client' )))

// Body Parser Middleware

app.use(express.json());

// Route middleware for users API

app.use('/users', usersRoute)

// Index route

app.get('/', (req, res) => {

    res.send("Invalid Endpoint");

});

1. Create models/schemas

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const config = require('../config/database');

// User Schema

const UserSchema = mongoose.Schema({

    firstname: {

        type: String,

        required: true

    },

    lastname: {

        type: String,

        required: true

    },

    email: {

        type: String,

        required: true

    },

    username: {

        type: String,

        required: true

    },

    password: {

        type: String,

        required: true

    }

});

const User = module.exports = mongoose.model('User', UserSchema);

module.exports.getUserById = function(id, callvack) {

    User.findById(id, callvack);

}

module.exports.getUserByUsername = function(user, callvack) {

   const query = {username: user};

    User.findOne(query, callvack);

}

1. Import user model into Users route file

const express = require('express');

const router = express.Router();

const passport = require('passport');

const jwt = require('jsonwebtoken');

const User = require('../models/user');

1. update add user function

// Register user

router.post('/register', (req, res) => {

    let newUser =  new User({

        firstname: req.body.firstname,

        lastname: req.body.lastname,

        email: req.body.email,

        username: req.body.username,

        password: req.body.password

    });

    User.addUser(newUser, (err, user) => {

        if(err) {

            res.jason({success: false, msg: 'Failed to register user'})

        }

        else {

            res.jason({success: true, msg: `${newUser.username} is now registered.`})

        }

    })

});

1. Create add user function in model and encrypt password using bcrypt

module.exports.addUser = function(newUser, callback){

    bcrypt.genSalt(10, (err, salt) => {

      bcrypt.hash(newUser.password, salt, (err, hash) => {

        if(err) throw err;

        newUser.password = hash;

        newUser.save(callback);

      });

    });

 }

1. Add passport middleware

// Passport middleware

app.use(passport.initialize());

app.use(passport.session());

1. Create passport.js

const JwtStrategy = require('passport-jwt').Strategy;

const ExtractJwt = require('passport-jwt').ExtractJwt;

const User = require('../models/user');

const config = require('./database');

module.exports = function(passport) {

    let opts = {};

    opts.jwtFromRequest = ExtractJwt.fromAuthHeaderWithScheme('jwt');

    opts.secretOrKey = config.secret;

    passport.use(new JwtStrategy(opts, (jwt\_payload, done) => {

        User.getUserById(jwt\_payload.\_id, (err, user) => {

            if(err) {

                return done(err, false);

            }

            if(user) {

                return done(null, user);

            }

            else {

                return done(null, false);

            };

        });

    }));

};

1. Add to app.js

require('./config/passport')(passport);

1. Work on authenticate

// Authenticate

router.post('/authenticate', (req, res, next) => {

    const username = req.body.username;

    const password = req.body.password;

    User.getUserByUsername(username, (err, user) => {

        if(err) {

            throw err;

        }

        // If user is not returned

        if(!user) {

            return res.json({success: false, msg: 'User not found'});

        }

        // If username is present, check password

        User.comparePassword(password, user.password, (err, isMatch) => {

            if(err) throw err;

            if(isMatch) {

                const token = jwt.sign(user.toJSON(), config.secret, {

                    expiresIn: 604800 // 1 week in seconds

                });

                res.json({

                    success: true,

                    token: 'JWT ' + token,

                    // Create oun user obj instead so password won't be included

                    user: {

                        id: user.\_id,

                        firstname: user.firstname,

                        lastname: user.lastname,

                        email: user.email

                    }

                });

            }

            else {

                return res.json({success: false, msg: 'Wrong password'});

            }

        })

    })

});

1. Create comparePassword
2. Test authenticate using postman
3. Make other routes protected