tuts+

CODE > NODE.JS

Authenticating Node.js Applications With Passport

by Agraj Mangal 11 Aug 2014

Difficulty: Intermediate Length: Long Languages: English

Node.js MongoDB JavaScript Front-End Back-End Web Development Databases Express

Implementing robust authentication strategies for any application can be a daunting task and Node.js applications are no exception to this.

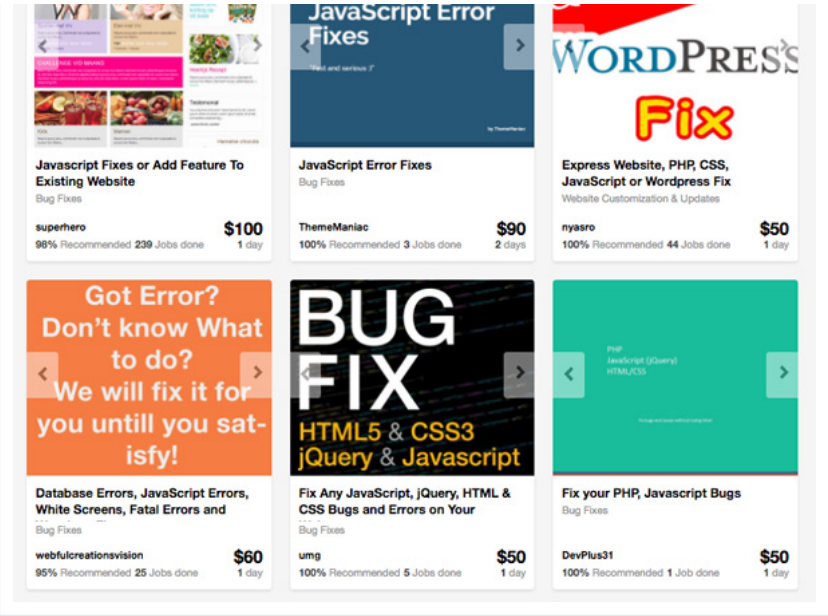
In this tutorial, we will develop a Node.js application from scratch and use a relatively new but very popular authentication middleware - Passport to take care of our authentication concerns.

> *Passport's documentation describes it as a "simple, unobtrusive authentication middleware for Node" and rightly so.*

By providing itself as a middleware, Passport does an excellent job at separating the other concerns of a web application from its authentication needs. It allows Passport to be easily configured into any Express-based web application, just like we configure other Express middleware such as logging, body-parsing, cookie-parsing, session-handling, etc.

This tutorial assumes a basic understanding of Node.js and Express framework and try to keep focus on authentication, although we do create a sample Express app from scratch and progress via adding routes to it and authenticating some of those routes.

If you need help with anything from debugging to new features, try working with some of the experienced JavaScript developers on Envato Studio.



JavaScript developers on Envato Studio

Authentication Strategies

Passport provides us with 140+ authentication mechanisms to choose from. You can authenticate against a local/remote database instance or use the single sign-on using OAuth providers for Facebook, Twitter, Google, etc. to authenticate with your social media accounts, or you can choose from an extensive list of providers which support authentication with Passport and provide a node module for that.

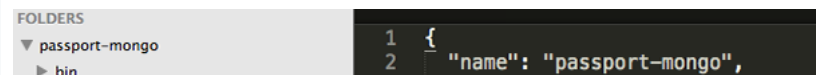
But worry not: You do not need to include any strategy/mechanism that your application does not need. All these strategies are independent of each other and packaged as separate node modules which are not included by default when you install Passport's middleware: `npm install passport`

In this tutorial, we will use the Local Authentication Strategy of Passport and authenticate the users against a locally configured Mongo DB instance, storing the user details in the database. For using the Local Authentication Strategy, we need to install the passport-local module: `npm install passport-local`

But wait: Before you fire up your terminal and start executing these commands, let's start by building a Express app from scratch and add some routes to it (for login, registration and home) and then try to add our authentication middleware to it. Note that we will be using Express 4 for the purposes of this tutorial, but with some minor differences Passport works equally well with Express 3, as well.

Setting Up the Application

If you haven't already, then go ahead and install Express & express-generator to generate a boilerplate application by simply executing `express passport-mongo` on the terminal. The generated application structure should look like this:



Let's remove some of the default functionality that we won't be making use of - go ahead and delete the `users.js` route and remove its references from the `app.js` file.

Adding Project Dependencies

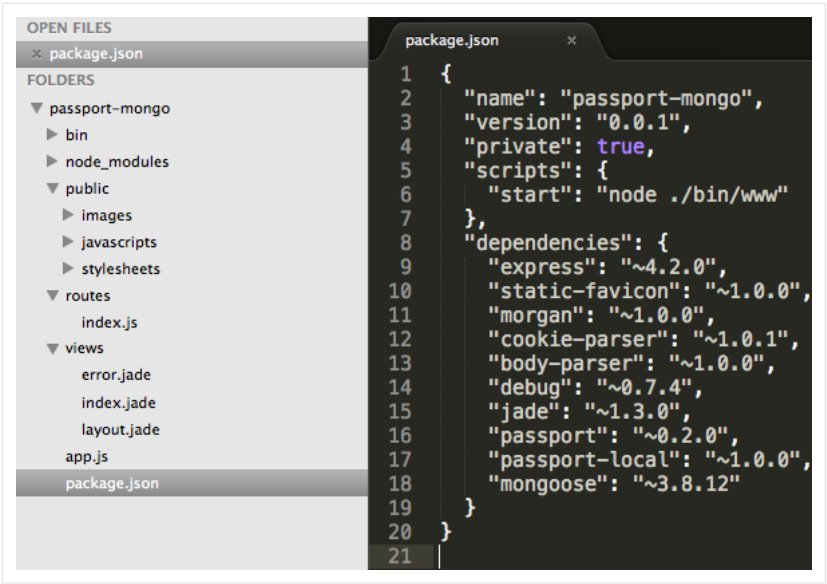
Open up `package.json` and add the dependencies for `passport` and `passport-local` module.

```
"passport": "~0.2.0",
"passport-local": "~1.0.0"
```

Since we will be saving the user details in MongoDB, we will use Mongoose as our object data modeling tool. Another way to install and save the dependency to `package.json` is by entering:

```
npm install mongoose --save
```

`package.json` should look like this:



Now, install all the dependencies and run the boilerplate application by executing `npm install && npm start`. It will now download and install all of the dependencies and will start the node server. You can check the basic Express app at http://localhost:3000/ but there is nothing much to see.

Very soon, we are going to change that by creating a full-fledged express app that asks for shows a registration page for a new user, the login of a registered user, and authenticates the registered user by using Passport.

Creating Mongoose Model

Since we will be saving the user details in Mongo, let's create a User Model in Mongoose and save that in `models/user.js` in our app.

```
var mongoose = require('mongoose');

module.exports = mongoose.model('User',{
        username: String,
    password: String,
    email: String,
    gender: String,
    address: String
});
```

Basically, we are creating a Mongoose model using which we can perform CRUD operations on the underlying database.

Configuring Mongo

If you do not have Mongo installed locally then we recommend that you use cloud database services such as Modulus or MongoLab. Creating a working MongoDB instance using these is not only free but is just a matter of few clicks.

After you create a database on one of these services, it will give you a database URI like `mongodb://<dbuser>:<dbpassword>@novus.modulusmongo.net:27017/<dbName>` which can be used to perform CRUD operations on the database. It's a good idea to keep the database configuration in a separate file which can be pull up as and when needed. As such, we create a node module `db.js` which looks like:

```
module.exports = {
  'url' : 'mongodb://<dbuser>:<dbpassword>@novus.modulusmongo.net:27017/<dbName>'
}
```

If you're like me, you are using a local Mongo instance then it's time to start the `mongod` daemon and the `db.js` should look like

```
module.exports = {
  'url' : 'mongodb://localhost/passport'
}
```

Now we use this configuration in `app.js` and connect to it using Mongoose APIs:

```
var dbConfig = require('./db.js');
var mongoose = require('mongoose');
mongoose.connect(dbConfig.url);
```

Configuring Passport

Passport just provides the mechanism to handle authentication leaving the onus of implementing session-handling ourselves and for that we will be using express-session. Open up `app.js` and paste the code below before configuring the routes:

```
// Configuring Passport
var passport = require('passport');
var expressSession = require('express-session');
app.use(expressSession({secret: 'mySecretKey'}));
app.use(passport.initialize());
app.use(passport.session());
```

This is needed as we want our user sessions to be persistent in nature. Before running the app, we need to install express-session and

add it to our dependency list in `package.json`. To do that type `npm install --save express-session`

Serializing and Deserializing User Instances

Passport also needs to serialize and deserialize user instance from a session store in order to support login sessions, so that every subsequent request will not contain the user credentials. It provides two methods `serializeUser` and `deserializeUser` for this purpose:

```
passport.serializeUser(function(user, done) {
  done(null, user._id);
});

passport.deserializeUser(function(id, done) {
  User.findById(id, function(err, user) {
    done(err, user);
  });
});
```

Using Passport Strategies

We will now define Passport's strategies for handling **login** and **signup**. Each of them would be an instance of the **Local Authentication Strategy** of Passport and would be created using the `passport.use()` function. We use connect-flash to help us with error handling by providing flash messages which can be displayed to user on error.

Login Strategy

The login strategy looks like this:

```
// passport/login.js
passport.use('login', new LocalStrategy({
    passReqToCallback : true
  },
  function(req, username, password, done) {
    // check in mongo if a user with username exists or not
    User.findOne({ 'username' :  username },
      function(err, user) {
        // In case of any error, return using the done method
        if (err)
          return done(err);
        // Username does not exist, log error & redirect back
        if (!user){
          console.log('User Not Found with username '+username);
          return done(null, false,
                req.flash('message', 'User Not found.'));
        }
        // User exists but wrong password, log the error
        if (!isValidPassword(user, password)){
          console.log('Invalid Password');
          return done(null, false,
              req.flash('message', 'Invalid Password'));
        }
        // User and password both match, return user from
        // done method which will be treated like success
        return done(null, user);
      }
    );
}));
```

The first parameter to `passport.use()` is the **name** of the strategy which will be used to identify this strategy when applied later. The second parameter is the **type** of strategy that you want to create, here we use the username-password or the LocalStrategy. It is to be noted that by default the LocalStrategy expects to find the user credentials in `username` & `password` parameters, but it allows us to use any other named parameters as well. The `passReqToCallback` config variable allows us to access the `request` object in the callback, thereby enabling us to use any parameter associated with the request.

Next, we use the Mongoose API to find the User in our underlying collection of Users to check if the user is a valid user or not. The last parameter in our callback : `done` denotes a useful method using which we could signal success or failure to Passport module. To specify

failure either the first parameter should contain the error, or the second parameter should evaluate to `false`. To signify success the first parameter should be `null` and the second parameter should evaluate to a `truthy` value, in which case it will be made available on the `request` object

Since passwords are inherently weak in nature, we should always encrypt them before saving them to the database. For this, we use bcrypt-nodejs to help us out with encryption and decryption of passwords.

```
var isValidPassword = function(user, password){
  return bCrypt.compareSync(password, user.password);
}
```

If you are feeling uneasy with the code snippets and prefer to see the complete code in action, feel free to browse the code here.

Registration Strategy

Now, we define the next strategy which will handle registration of a new user and creates his or her entry in our underlying Mongo DB:

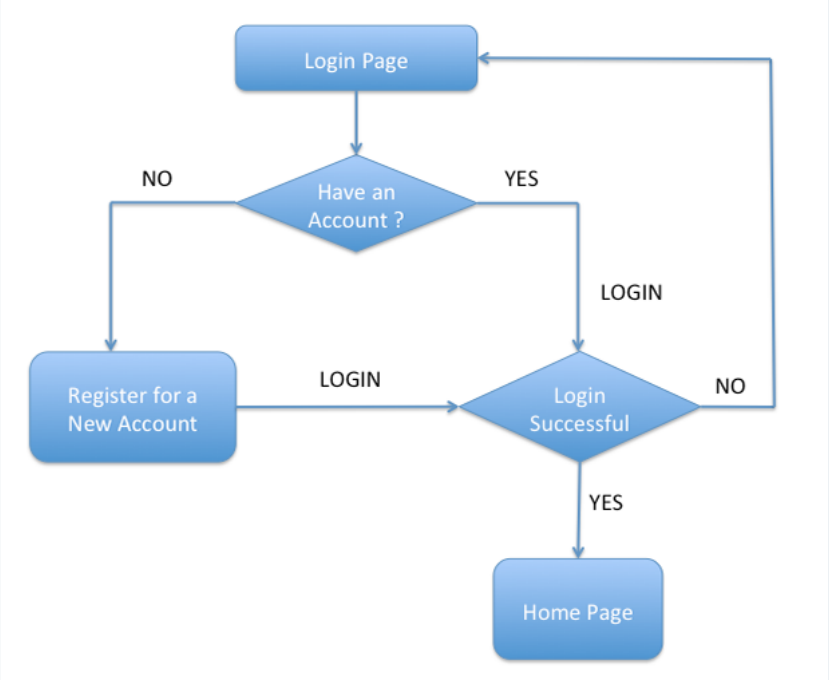
```
passport.use('signup', new LocalStrategy({
    passReqToCallback : true
  },
  function(req, username, password, done) {
    findOrCreateUser = function(){
      // find a user in Mongo with provided username
      User.findOne({'username':username},function(err, user) {
        // In case of any error return
        if (err){
          console.log('Error in SignUp: '+err);
          return done(err);
        }
        // already exists
        if (user) {
          console.log('User already exists');
          return done(null, false,
             req.flash('message','User Already Exists'));
        } else {
          // if there is no user with that email
          // create the user
          var newUser = new User();
          // set the user's local credentials
          newUser.username = username;
          newUser.password = createHash(password);
          newUser.email = req.param('email');
          newUser.firstName = req.param('firstName');
          newUser.lastName = req.param('lastName');

          // save the user
          newUser.save(function(err) {
            if (err){
              console.log('Error in Saving user: '+err);
              throw err;
            }
            console.log('User Registration succesful');
            return done(null, newUser);
          });
        }
      });
    };

    // Delay the execution of findOrCreateUser and execute
    // the method in the next tick of the event loop
    process.nextTick(findOrCreateUser);
  });
);
```

Here, we again use the Mongoose API to find if any user with the given username already exists or not. If not, then create a new user and saves the user information in Mongo. Else return the error using the `done` callback and flash messages. Note that we use `bcrypt-nodejs` for creating the hash of the password before saving it:

```
// Generates hash using bCrypt
var createHash = function(password){
 return bCrypt.hashSync(password, bCrypt.genSaltSync(10), null);
}
```

Creating Routes

If we were to see a birds eye view of our application, it would look like:



We now define our routes for the application in the following module which takes the instance of Passport created in `app.js` above. Save this module in `routes/index.js`

```
01 module.exports = function(passport){
02
03   /* GET login page. */
04   router.get('/', function(req, res) {
05     // Display the Login page with any flash message, if any
06     res.render('index', { message: req.flash('message') });
07   });
08
09   /* Handle Login POST */
10   router.post('/login', passport.authenticate('login', {
11     successRedirect: '/home',
12     failureRedirect: '/',
13     failureFlash : true
14   }));
15
16   /* GET Registration Page */
17   router.get('/signup', function(req, res){
18     res.render('register',{message: req.flash('message')});
19   });
20
21   /* Handle Registration POST */
22   router.post('/signup', passport.authenticate('signup', {
23     successRedirect: '/home',
24     failureRedirect: '/signup',
25     failureFlash : true
26   }));
27
28   return router;
29 }
```

The most important part of the above code snippet is the use of `passport.authenticate()` to delegate the authentication to `login` and `signup` strategies when a HTTP `POST` is made to `/login` and `/signup` routes respectively. Note that it is not mandatory to name the strategies on the route path and it can be named anything.

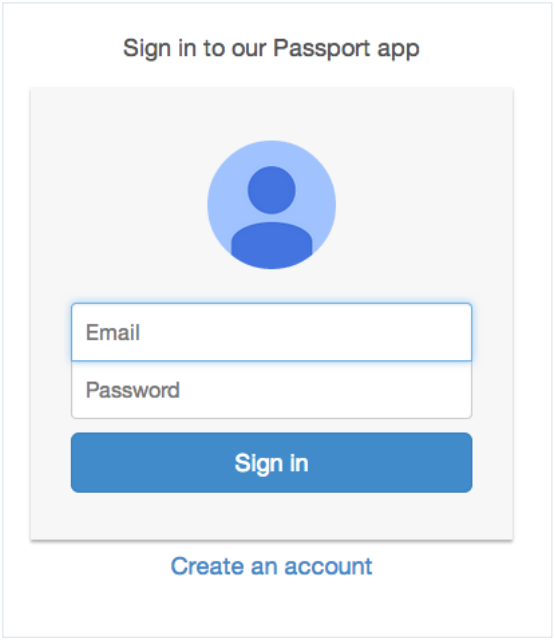
Creating Jade Views

Next, we create the following two views for our application:

1. `layout.jade` contains the basic layout & styling information
2. `index.jade` contains the login page containing the login form and giving option to create a new account

```
extends layout

block content
  div.container
    div.row
      div.col-sm-6.col-md-4.col-md-offset-4
        h1.text-center.login-title Sign in to our Passport app
          div.account-wall
            img(class='profile-img', src='https://lh5.googleusercontent.com/-b0-k99FZlyE/AAAAAAAAAAI/AAAAAAAAAAA/eu7opA4byxI/photo.jpg?
            form(class='form-signin', action='/login', method='POST')
              input(type='text', name='username' class='form-control', placeholder='Email',required, autofocus)
              input(type='password', name='password' class='form-control', placeholder='Password', required)
              button(class='btn btn-lg btn-primary btn-block', type='submit') Sign in
              span.clearfix
          a(href='/signup', class='text-center new-account') Create an account
          #message
          if message
            h1.text-center.error-message #{message}
```

Thanks to Bootstrap, our Login page now looks like



We need two more views for registration details and for the home page of the application:

1. `register.jade` contains the registration form
2. `home.jade` says hello and shows logged in user's details

If you are unfamiliar with Jade, check out the documentation.

Implementing Logout Functionality

Passport, being a middleware, is permitted to add certain properties and methods on request and response objects and it makes proper use of it by adding a very handy `request.logout()` method which invalidates the user session apart from other properties.

```
/* Handle Logout */
router.get('/signout', function(req, res) {
  req.logout();
  res.redirect('/');
});
```


Protecting Routes

Passport also gives the ability to protect access to a route which is deemed unfit for an anonymous user. This means that if some user tries to access http://localhost:3000/home without authenticating in the application, he will be redirected to home page by doing

```
/* GET Home Page */
router.get('/home', isAuthenticated, function(req, res){
  res.render('home', { user: req.user });
});

// As with any middleware it is quintessential to call next()
// if the user is authenticated
var isAuthenticated = function (req, res, next) {
  if (req.isAuthenticated())
    return next();
  res.redirect('/');
}
```

Conclusion

Passport is not the only player in this arena when its comes to authenticating Node.js applications and there exists alternatives like EveryAuth but the modularity, flexibility, community support and the fact that its just a middleware makes Passport definitely a much better choice.

For a detailed comparison between the two, here is an interesting and informative perspective from the developer of Passport himself.

If you want to see what else you can do with Node.js, check out the range of Node.js items on Envato Market, from a responsive AJAX contact form to a URL shortener, or even a database CRUD generator.

Agraj Mangal

Agraj Mangal is a full stack developer working for more than 6 years. He is majorly focused on Java, J2EE, OSGi based enterprise and web applications, but is equally inclined towards client side development using HTML5, JS & CSS3. He is always looking out to learn and experiment with new technologies.

agrajm

View on Github

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by native

81 Comments **Tuts+ Hub** **Login**

Recommend 5 **Share** **Sort by Best**

Join the discussion…

MaxArt • 2 years ago
I started studying node.js recently and this is exactly the right article at the right time for me.
Clear and concise.
Thank you!
10 • Reply • Share ›

Agraj ➚ MaxArt • 2 years ago
Thanks. Stay tuned for another Passport article coming up which deals with authenticating with social media accounts such as Facebook/Twitter etc.
10 • Reply • Share ›

Francisco Presencia ➚ Agraj • 2 years ago
Any update on this? (:
• Reply • Share ›

Agraj ➚ Francisco Presencia • 2 years ago
Hi Francisco, the second article dealing with social authentication can be found here - http://code.tutsplus.com/artic...
• Reply • Share ›

Agraj ➚ Francisco Presencia • 2 years ago
Hi, I've already submitted another article using Passport where we authenticate with Twitter/Facebook and its in the normal processing queue for the articles and should come out by February end.
• Reply • Share ›

HelloNode ➚ Agraj • 2 years ago
Please upload the next article in the series quickly.
• Reply • Share ›

Agraj ➚ HelloNode • 2 years ago
And the second article is up. Please check it out here - http://code.tutsplus.com/artic...
1 • Reply • Share ›

MaxArt ➚ Agraj • 2 years ago
That would be great!
• Reply • Share ›

woojoo666 • 2 years ago

very nice article, just noticed by comparing the tutorial and the source that in the "Serializing and Deserializing User Instances", you forgot to mention that the code is supposed to be put in a new file passport/init.js (with some boilerplate code), and that we should call

```
var initPassport = require('./passport/init');
initPassport(passport);
```

in app.js

6 • Reply • Share ›

JR Halchak ➚ woojoo666 • 2 years ago

Thank you so much, I was getting frustrated. For some reason, no node tutorial I've read consistently tells you where they put things / how they structure their apps.

4 • Reply • Share ›

Agraj ➚ JR Halchak • 2 years ago

Thank you for the kind words JR Halchak. We have published a second article in the same authentication series. You might find that useful as well

• Reply • Share ›

herbWat • 2 years ago

first off great artice, thank you
the only thing i noticed is that when signing in it says email in the top field, but its expecting the username or am i getting something wrong there ? :p

4 • Reply • Share ›

Justin Case ➚ herbWat • 2 years ago

It's a typo in the /views/index.jade code where it says placeholder. See my other reply below.

• Reply • Share ›

Guest ➚ Justin Case • 2 years ago

Thank you for the kind words. We have published a second article in the same authentication series. You might find that useful as well

• Reply • Share ›

pushplaybang • 2 years ago

while some of this is great, it does seem somewhat incomplete, reading the source was cool, though it also feels a little inconsistent. good place to get started though. cheers.

2 • Reply • Share ›

Steve • 2 years ago

Excellent Tut and Codebase! I am seeing a problem with express-session and once logged in and I do a Get "/", I am not logged in any more. In the network trace there are no cookies, but maybe its my fault?

express-session deprecated undefined resave option; provide resave option api.js:76:9
express-session deprecated undefined saveUninitialized option; provide saveUninitialized option api.js:76:9

2 • Reply • Share ›

pushplaybang ➚ Steve • 2 years ago

same error.... pass in resave:true, saveUninitialized:true with your secret key :)

1 • Reply • Share ›

HelloNode ➚ pushplaybang • 2 years ago

I am also getting the same error. And even after passing resave:true, saveUninitialized:true it does not work.

• Reply • Share ›

Tony Brown • 2 years ago

excellent article

2 • Reply • Share ›

dapinitial • 2 years ago

Dank

2 • Reply • Share ›

Rohil Shah • a year ago

is there anyway that i can add a new column? like i want to handle a search history for every user. so can i input the searched keywords into the database after the user logs in and searches for a keyword?

1 • Reply • Share ›

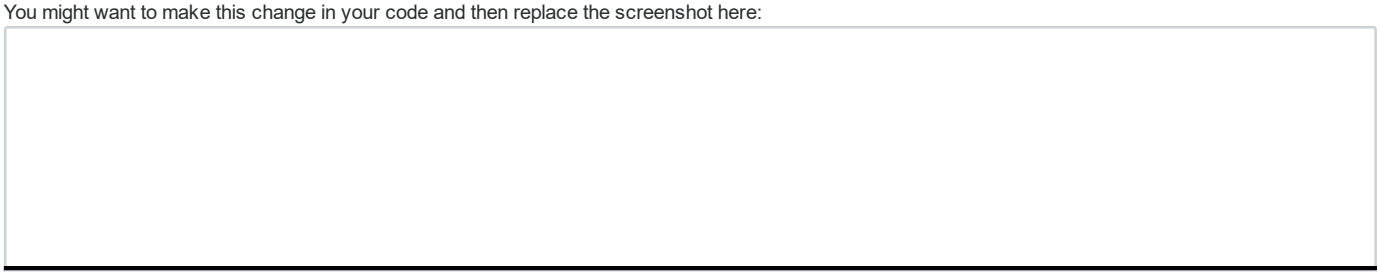
Justin Case • 2 years ago

Fantastic article and code. :) This really helps a lot for us noobs.

Btw, /views/index.jade has placeholder='Email' but should have placeholder='Username' instead. In fact, I slightly modified the code here and on the signup page...
input(type='text', name='username' class='form-control', placeholder='Username', value='Username', required, onfocus='this.select()', autofocus)

[This will put the text Username in the field and highlight it with the autofocus selecton. Otherwise, the default style of the input box hides the purpose of the first field.]

You might want to make this change in your code and then replace the screenshot here:



see more

1 • Reply • Share ›

HelloNode • 2 years ago

What is the procedure to check whether a user is logged in or not?

1 • Reply • Share ›

jvrunion • 2 years ago

nice article!

1 • Reply • Share ›

Ernst Maurer • a month ago

tried to use this as my first sample project for node, cloned from github,
server started , but no one function works, login or signup just do nothing.
On server side , I only see that "something" is happening , but (using breakpoints) runflow does go to those function never

is this really working? and tested?. (tried with local and remote mongo instances)

• Reply • Share ›

Ben • 2 months ago

Can you please update the code listings to explicitly indicate what files you are putting what code in? I find this is a huge problem for just about every tutorial I encounter. It just makes it a lot clearer! Please?

• Reply • Share ›

Wolf Of the Dead • 5 months ago

please is there anyone here? i need help with this error
Error: Cannot find module './passport/init'

• Reply • Share ›

Wolf Of the Dead • 5 months ago

wait a sec? did everyone forget to mention that you have to define passport and localstrategy? or i am the one that does not know how to program?

• Reply • Share ›

Wolf Of the Dead • 5 months ago

please where will the Serializing and Deserializing User Instances go to, to passpoer.js or app.js?

• Reply • Share ›

Rajasekaran Murugesan • 7 months ago

is this avail in github or avail in any source? because, when i want use newUser.local.firstname = req.param('firstname'); with email and password means, i got error 'Folk unexpected token "indent" at Parser.parseExpr'..whats this error i dont know how to solve

• Reply • Share ›

Rajasekaran Murugesan • 7 months ago

is this avail in github or avail in any source? because, when i want use newUser.local.firstname = req.param('firstname'); with email and password means, i got error 'Folk unexpected token "indent" at Parser.parseExpr'..whats this error i dont know how to solve

• Reply • Share ›

Benji Lamalice • 9 months ago

excellent ! thanks you for this article, exactly what i want
Reply • Share ›

sudeep • 10 months ago
thanks
Reply • Share ›

Nikk Wong • a year ago
Why do we use process.nextTick(findOrCreateUesr) ?
Reply • Share ›

void • a year ago
Thank you so much for this tuto ! That's exactly what I needed as a noob :)
Reply • Share ›

cysi • a year ago
Hi, sorry for coming back to this old tuto but it really is great.
I'd like to put a passport authentification before giving access to another node app,
are there any solutions to do that ?
Reply • Share ›

getItRight • a year ago
This is a very detailed and interesting to read. Thanks man!!
Reply • Share ›

David Zhu • a year ago
Thanks for making this!
Reply • Share ›

Zeeshan Ahmad • a year ago
The app gives me this:

TypeError: undefined is not a function
at module.exports (.....\passport-mongo\routes\index.js:25:33)

this is line 25:
router.post('/login', passport.authenticate('login', {
successRedirect: '/home',
failureRedirect: '/',
failureFlash : true
}));

when I do console.log(passport.authenticate); it comes out as 'undefined' on the console.
Reply • Share ›

Dylan ➚ Zeeshan Ahmad • a year ago
Hi had the same issue, I was copying the code into another project. The problem for me was my original app.js, I copied over all the code (not just portions) into my app.js, and it resolved my problem.
Reply • Share ›

A_A ➚ Zeeshan Ahmad • a year ago
Same error, cant find a solution anywhere, any ideas?
Reply • Share ›

Benji Lamalice ➚ A_A • 9 months ago
don't forget to pass "passport" instance in the function call like this :
var routes = require('./routes/index')(passport);
Reply • Share ›

Jonathan Souied • a year ago
Nice article :)

To deal with permissions just replace simple function : var isAuthenticated = function(req, res, next) in route index.js by :

var haveGoodRole = function(role) {
return function(req, res, next) {
if (req.session.user && req.session.user.role === role)
next();

```
else
res.redirect('/');
}
}
}
```

You can customize it by arrange user.role in Array of role and search for role in the field. Must have for multi-permissions app.

Reply • Share ›

Faiza Qamar • a year ago

This authentication didn't work for the "BACK" button of browser. After signing out, pressing "Back button of browser" and m back to restricted area... Any solution?

Reply • Share ›

dohkoo • a year ago

This article has a good start but is incomplete -- half way through it just lists code but doesn't tell where it should go or how it fits into the whole process. I had to go to the code repository to find out where you put it or what files should be in what folder. Thanks for putting it together though

Reply • Share ›

Javier Muñoz de Baena Bertran • a year ago

Excellent tutorial, thank you!

Reply • Share ›

Supraja • 2 years ago

Hi , very good article and I got it working.Could you help me out in fetching user details from the session and to display them in the view ? Im using Angularjs(html and not jade)

Reply • Share ›

Rajan Pal • 2 years ago

Basic Question: Is it possible to use passport authentication with simple vanilla nodejs or it can be built only on top of other frameworks like express?

Reply • Share ›

Sergio Andrés Ñustes • 2 years ago

More easy, imposible, thank you!

Reply • Share ›

Nyreen • 2 years ago

Can I ask is there a way to specify which collection in the database provided should be searched for the username and password for the log-in part?

Reply • Share ›

Load more comments