PASSPORT

"passport" module

Passport is authentication middleware for Node.js. It is designed to serve a singular purpose: authenticate requests

Passport

exports : Authenticator Object

Authenticator.prototype

- **use(name, strategy)**
  - Utilize the given `strategy` with optional `name`, overridding the strategy's default name. by utilizing it means that this function registers the strategy in _strategies object
- **unuse(name)**
- **initialize(options)**
  - uses and returns this._framework.initialize middleware function to initialize the passport for use.
  - This middleware must be in use by the Connect/Express application for Passport to operate.
- **authenticate(strategy,option,callback)**
  - uses and returns ***this._framework.authenticate(this,strategy,...)*** middleware function
  - Middleware will authenticate a request using the given `strategy` name, with optional `options` and `callback`.
- **serializeUser(fn,req,done)**
  - if fn which is passed as function than serializeUser function registers 'fn' to the list _serializers
  - else the passed fn will not be an actual function but a 'User' thus serializeUser function travers the list _serializers run all the functions listed for this 'user' (this part is for internal implemtation)
  - this functions are used ***inside the session manager (_sn) for login and logout functionality***
- **deserializeUser(fn,req,done)**

Prototype

- _key = 'passport';
- **_strategies = {};**
  - name: passport-strategy
    - 'session' : SessionStrategy
- **_serializers = []**
  - list of serializer functions
- **_deserializers = [];**
  - list of deserializer functions
- **_framework = {**
  authenticate function
  initialize function
  }
- **_sm (session manager)**

managing session (server-side)

Middleware Framework functions

Session Manager(options, serializeUser)

_key = options.key or "passport"
_serializeUser = serializeUser

Prototype
- login(req,user,cb)
  - uses serializeUser to save the user into session
    - *req.session[this._key].user*
- logout(req,cb)
  - deletes req.session[this._key].user

Frameworks

this._framework.initialize

this._framework.authenticate(
passport,
name, // strategy name list
options,
callback
)

- If sessions are being used, this middleware must be in use by the Connect/Express application for Passport to operate. If the application is entirely stateless (not using sessions), this middleware is not necessary, but its use will not have any adverse
- Note that additional middleware is required to persist login state, so must use the `connect.session()` middleware _before_ `passport.initialize()`.
  - app.use(connect.session({ secret: 'keyboard cat' }));
  - then
  - app.use(passport.initialize());
- internally this middleware takes req and sets few properties to it namely
  - req._userProperty
  - req._passport = {}
  - req._passport.instance = passport (if not provided equals to null)

```
* Authenticates requests.
*
* Applies the `name`ed strategy (or strategies) to the incoming request, in
* order to authenticate the request.  If authentication is successful, the user
* will be logged in and populated at `req.user` and a session will be
* established by default.  If authentication fails, an unauthorized response
* will be sent.
*
* Options:
*   - `session`        Save login state in session, defaults to _true_
*   - `successRedirect`  After successful login, redirect to given URL
*   - `successMessage`  True to store success message in
*                       req.session.messages, or a string to use as override
*                       message for success.
*   - `successFlash`  True to flash success messages or a string to use as a flash
*                       message for success (overrides any from the strategy itself).
*   - `failureRedirect`  After failed login, redirect to given URL
*   - `failureMessage`  True to store failure message in
*                       req.session.messages, or a string to use as override
*                       message for failure.
*   - `failureFlash`  True to flash failure messages or a string to use as a flash
*                       message for failures (overrides any from the strategy itself).
*   - `assignProperty`  Assign the object provided by the verify callback to given property
*
```

- returns authenticate(req,res,next) which
  - sets some important properties regarding the user state to the req object
    - req.login = _sm.logIn || req.logIn => sets property to user
    - req.logout = sm.logOut || req.logOut => sets property to null
    - req.isAuthenticated = true if the request is authenticated (checks the property)
    - req._sessionManager = _sm
  - After setting all the properties to req object, it iterates through name - list of strategy to authenticate the request against. if the list is not passed and a single strategy-name is passed it is converted to a list with single strategy name.
  - for i in range(length(name))
    - strategy = name[0]
    - it first augments a strategy object with success, fail , redirect and error functions which when the some
    - strategy.success = function(user,info){...}
      - calls req.login interally to populate the user in req.user
    - strategy.fail = function(challenge,status){...}
    - strategy.redirect = function(challenge,status){...}
    - strategy.error = function(err){...}
    - strategy.pass = function(){next()}
    - **finally stategy.authenticate(req,options) is called**
    - which will invoke the authenticate function of the particular strategy
    - the first strategy which succesfully authenticates i.e. calls strategy.success this loop stops and calls the next().

Diagram Key

- node - module
- prototype
- index
- object

authenticate(req,option) connects this two packages

Strategy

"passport-strategy"
"passport-outh2"
"passport-google-outh20"
modules

Strategy.prototype

@abstract
authenticate(req,option)

this function must be overriden by subclasses or else throws error

Passport-Strategy

Exports : Strategy

Oauth2

oauth2 api allows users to authenticate against Oauth Providers.
passport-oauth uses this package to authetication request by delegating to service providers in **OAuth2Strategy.prototype.authenticate** function.

Passport-outh2

creates an Instance of OAuth2Strategy which authenticates requests using OAuth 2.0 framework.

Imports the Strategy

OAuth2Strategy(options,varify)

Options:
- 'authorizationURL' - URL used to obtain an authorization grant
- 'tokenURL' - URL used to obtain an authorization token
- 'clientID' - Identifies Client to Service Provider
- 'clientSecret' - Establish ownership of the client identifier (client ID)
- 'callbackURL' - URL to redirect the user after Authorization.

Varify callback:
function(accesstoken ,
refreshtoken,
profile,
done) {.....}
The varify callback is responsible for finding or creating the user, and invoking 'done(err,user,info)'

- done(false,null) if the user not found
- done(null, false) if the user found, but incorret password.
- done(null, {authenticated_user}) if userfound and passwword match.

the done() function is then used to pass the {authenticated_user} to the serializeUser() function.

Populates the Strategy imported with below

- name = "oauth2"
- _varify = varify callback passed in construction function
- _scope = options.scope
- _callbackURL = options.callbackURL

etc

inherits(Oauth2Strategy,Strategy)

OAuth2Strategy inherites from Strategy and implements the abstract authenticate function and add somme other abstract functions of its own.

- name = "oauth2"
- _varify = varify callback passed in construction function
- _scope = options.scope
- _callbackURL = options.callbackURL

etc

Prototype:
- **authenticate(req,option)**
  here the options includes the options passed in the construction function of OAuth2Strategy
- @abstract
  userProfile(accesstoken,done)
- @abstract
  authorizationParams(options)
- @abstract
  tokenParams(options)

returns - OAuth2Strategy

The authenticate(req,option) function checks if the req contains the authorization code or not

---------- if ----------

if req do not contain authorization code
(req.body.code == null) imples the request is for authentication , then it
gets and sets a few params = authorizationParams()

sets
- param.response_type = code
- param.redirect_uri = callbackURL
- param.scope = options.scope || this._scope;
- params.code_challenge = challenge;
- params.code_challenge_method = this._pkceMethod;

then it constructs the url witch the above parameters and redirect to it which will be to the third party authentication server for getting the authorization code.
it redirects using **redirect(url,status) which was setup by passport.authenticate() function**

---------- else ----------

if req contains the authorization code it implies that its a callback from the authorization server.
here it first varifies the state and then sets some params = tokenParams()

sets
- param.grant_type = "authorization code"
- params.redirect_uri = callbackURL (if not mentioned again in options it uses the old this._callbackURL.

here it sets grant_type to authorization code as it will use the authorization code and params to get varified to the third party server and get the accesstoken in return,
getOAuthAccessToken(code,params,
callbackFunction(accesstoken,refreshtoken,...)
)

once it the accesstoken from the authorization server , it will call the above callback function in which theres a function cllied loadUserProfile which will fetch the user profile against the accessToken.

_loadUserProfile(accessToken,
callback(err,profile){...})

once it receives the profile back from the authorization server the **loadUserProfile** function calls the **callback(err,profile)**

In this callback(err,profile)
if the err has occured it raises and error
and
if the profile has been successfully fetched
it calls **this._verify** which is a function we passed initially.
this function is used to get a check from user that the profile extracted is correct or do something with the profile before it is saved.
thus
_verify(req, accessToken, refreshToken, profile, verified);

here the verified is callback function which user calls after the check. it is the same done function we discussed before.
verified(err, user, info) where if the err is null and user is found(true) , the user and info is succesfully passed on **success(user,info)**.

this success funciton is setup by passport.authenticate function

Passport-google -outh2

Strategy (options, varify)

- name = "google"
- _varify = varify callback passed in construction function
- _scope = options.scope
- _callbackURL = options.callbackURL
- _userProfile ='https://www.googleapis.com/oauth2/v3/userinfo';

etc

inherits(Strategy,OAuth2Strategy)

Strategy (google-oauth2)
inherites from OAuth2Strategy and implements the inherited abstract functions UserProfile and authorizationParams

- name = "google"
- _varify = varify callback passed in construction function
- _scope = options.scope
- _callbackURL = options.callbackURL
- _userProfile ='https://www.googleapis.com/oauth2/v3/userinfo';
- etc

Prototype:
- authenticate(req,option)
  here the options includes the options passed in the construction function of OAuth2Strategy
- userProfile(accesstoken,done)
- authorizationParams(options)
- @abstract
  tokenParams(options)

this strategy constructor function takes options and a varify call back

it adds some extra options specific to google auth server to options

options.authorizationURL = 'https://accounts.google.com/o/oauth2/v2/auth';
options.tokenURL = https://www.googleapis.com/oauth2/v4/token';

after this it simly calls OAuth2Strategy.calls(this, options, varify)

which setups the "this" to contain all the fields setup-ed in OAuth

then it updates two fields
- **this.name = 'google'**
- **this._userProfileURL = 'https://www.googleapis.com/oauth2/v3/userinfo'**