COMP 2068
Advanced Web Programming

Lesson 8 – Authenticating Using Passport

Georgian

Objectives

1. Understanding Passport strategies
2. Integrating Passport into your users' MVC architecture
3. Using Passport's local strategy to authenticate users
4. An Overview of Passport OAuth strategies

Introducing Passport

- **Authentication** is a vital part of most web applications.
- Handling user registration and sign-in is an important feature, which can sometimes present a development overhead.
- Express, with its lean approach, lacks this feature, so, as is usual with node, an external module is needed.
- **Passport** is a Node.js module that uses the middleware design pattern to authenticate requests.
- It allows developers to offer various authentication methods using a mechanism called **strategies**, which allows you to implement a complex authentication layer while keeping your code clean and simple.
- Just as with any other Node.js module, before you can start using it in your application, you will first need to install it.

Installing Passport

- Passport uses different modules, each representing a different authentication strategy, but all of which depend on the base Passport module.
- To install the Passport base module in your application's modules folders type the following into a terminal window or command prompt:
  - `npm install passport --save`
- This will add Passport to the package.json file as a project dependency.

Configuring Passport

1. In the app.js file add the following line near other package registration statements:
   - `var passport = require('passport');`

2. Then before your route declarations add the following lines:
   - `app.use(passport.initialize());`
   - `app.use(passport.session());`

❖ The **`passport.initialize()`** middleware, which is responsible for bootstrapping the Passport module and the **`passport.session()`** middleware, which is using the **Express** session to keep track of your user's session.

Understanding Passport strategies

- To offer its various authentication options, **Passport** uses separate modules that implement different **authentication strategies**.
- Each module provides a different authentication **method**, such as **username/password** authentication and **Oauth** authentication.
- So, in order to offer Passport-supported authentication, you'll need to install and configure the strategies modules that you'd like to use.

Using Passport's local strategy

- Passport's local strategy is a **Node.js** module that allows you to implement a **username/password** authentication mechanism.
- You'll need to install it like any other module and configure it to use your User Mongoose model.
- Let's begin by installing the local strategy module.
  - `npm install passport-local --save`

Configuring Passport's Local Strategy

```
1  //var passport = require('passport');
2  var LocalStrategy = require('passport-local').Strategy;
3
4  // Import the User Model
5  var User = require('../models/user');
6
7  module.exports = function (passport) {
8
9    // serialize user
10   passport.serializeUser(function (user, done) {
11     done(null, user);
12   });
13
14   // deserialize user
15   passport.deserializeUser(function (id, done) {
16     User.findById(id, function (err, user) {
17       done(err, user);
18     });
19   });
20
```

Configuring Passport's Local Strategy (Cont'd)

```
21  passport.use('local-login', new LocalStrategy({
22    passReqToCallback: true
23  },
24
25    function (req, username, password, done) {
26
27      // asynchronous process
28      process.nextTick(function () {
29        User.findOne({
30          'username': username,
31        }, function (err, user) {
32          if (err) {
33            return done(err);
34          }
35
36          // no valid user found
37          if (!user) {
38            // third parameter is a flash warning message
39            return done(null, false, req.flash('loginMessage', 'Incorrect username'));
40          }
41
42          // no valid password entered
43          if (!user.validPassword(password)) {
44            return done(null, false, req.flash('loginMessage', 'Incorrect password'));
45          }
46
47          // everything ok - proceed with login
48          return done(null, user);
49        });
50      });
51    }));
```

Configuring Passport's Local Strategy (Cont'd)

```
53      // Configure registration local strategy
54      passport.use('local-registration', new LocalStrategy({
55          passReqToCallback : true
56      },
57      function(req, username, password, done) {
58
59          // asynchronous process
60          process.nextTick(function() {
61              // if the user is not already logged in:
62              if (!req.user) {
63                  User.findOne({ 'username' : username },
64                  function(err, user) {
65                      // if errors
66                      if (err) {
67                        return done(err);
68                        }
69                      // check email
70                      if (user) {
71                          return done(null, false, req.flash('registerMessage','The username is already taken.'));
72                      }
73                      else {
```

Configuring Passport's Local Strategy (Cont'd)

```
74                          // create the user
75                          var newUser = new User(req.body);
76                          newUser.password = newUser.generateHash(password);
77                          newUser.provider = 'local';
78                          newUser.created = Date.now();
79                          newUser.updated = Date.now();
80                          newUser.save(function(err) {
81                              if (err) {
82                                throw err;
83                                }
84                              return done(null, newUser);
85                          });
86                      }
87                  });
88              } else {
89                  // everything ok, register user
90                  return done(null, req.user);
91              }
92          });
93      }));
94  }
```

Configuring Passport's Local Strategy (Cont'd)

- The preceding code begins by requiring the **Passport** module, the local strategy module's Strategy object, and your User Mongoose model.
- Then, you register the strategy using the `passport.use()` method that uses an instance of the **LocalStrategy** object.
- Notice how the **LocalStrategy** constructor takes a callback function as an argument.
- It will later call this callback when trying to authenticate a user.

- The `passport.serializeUser()` and `passport.deserializeUser()` methods are used to define how Passport will handle user **serialization**.
- These allow the user's data to be saved and retrieved from a **session** store.
- When a user is authenticated, Passport will save its **_id** property to the session.
- Later on when the user object is needed, Passport will use the **_id** property to grab the user object from the database.

Creating a User model

```
1  // Import mongoose and bcrypt
2  var mongoose = require('mongoose');
3  var bcrypt = require('bcrypt-nodejs');
4  var Schema = mongoose.Schema;
5
6  var UserSchema = new Schema({
7    username: String,
8    password: String,
9    email: String,
10   displayName: String,
11   salt: String,
12   provider:String,
13   providerId: String,
14   providerData: {},
15   created: Number,
16   updated: Number
17 }, {
18   collection: 'userInfo'
19 });
20
21 // Generating a Hash
22 UserSchema.methods.generateHash = function(password) {
23     return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null);
24 };
25
26 // Checking if password is valid
27 UserSchema.methods.validPassword = function(password) {
28     return bcrypt.compareSync(password, this.password);
29 };
30
31 module.exports = mongoose.model('User', UserSchema);
```

Creating the authentication views

- Just as with any web application, you will need to have **Registration** and **Login** pages in order to handle user authentication.
- We'll create those views using the EJS template engine, so in your Views folder, create a **register.ejs** and **login.ejs**.

Creating the Login View (`login.ejs`)

```
<% include partials/header.ejs %>
      <main class="container">
      <% if (messages.length > 0) { %>
         <div class="alert alert-danger"><%= messages %></div>
       <% } %>
       <h1>Please Login</h1>
       <div class="col-sm-4">
            <form method="post">
                 <fieldset class="form-group">
                     <label>Username:</label>
                     <input type="text" class="form-control" name="username" required autofocus/>
                  </fieldset>
                  <fieldset class="form-group">
                     <label>Password:</label>
                     <input type="password" class="form-control" name="password" required/>
                     or <a href="/register">Register</a>
                 </fieldset>
                 <fieldset class="form-group text-right">
                     <input type="submit" class="btn btn-success" value="Log In"/>
                </fieldset>
             </form>
          </div>
    </main>
<% include partials/footer.ejs %>
```

Creating the Login View (`register.ejs`)

```
<% include partials/header.ejs %>
    <!-- Render the Registration form  -->
    <main class="container">
    <!-- Display flash messages  -->
    <% if (messages.length > 0) { %>
        <div class="alert alert-danger"><%= messages %></div>
      <% } %>
        <h1>User Registration</h1>
        <div class="col-sm-4">
            <form method="post">
                <fieldset class="form-group">
                    <label for="email">Email: *</label>
                    <input name="email" type="email" class="form-control" required autofocus />
                </fieldset>
                <fieldset class="form-group">
                    <label for="username">Username: *</label>
                    <input name="username" type="text" class="form-control" required />
                </fieldset>
                <fieldset class="form-group">
                    <label for="password">Password: *</label>
                    <input name="password" type="password" class="form-control" required />
                </fieldset>
                <fieldset class="form-group">
                    <label for="displayName">Display Name: *</label>
                    <input name="displayName" type="text" class="form-control" required />
                </fieldset>
                <fieldset class="form-group text-right">
                    <input type="submit" class="btn btn-success" value="Submit"/>
                </fieldset>
            </form
        </div>
    </main>
<% include partials/footer.ejs %>
```

Adding Authentication Routing

```
1  var express = require('express');
2  var passport = require('passport');
3  var router = express.Router();
4
5  var User = require('../models/user');
6
7  /* Render home page. */
8  router.get('/', function (req, res, next) {
9      res.render('index', {
10         title: 'Home',
11         displayName: req.user ? req.user.displayName : ''
12     });
13 });
14
15 /* Render Login page. */
16 router.get('/login', function (req, res, next) {
17     if (!req.user) {
18         res.render('login', {
19             title: 'Login',
20             messages: req.flash('loginMessage'),
21             displayName: req.user ? req.user.displayName : ''
22         });
23     }
24     else {
25         return res.redirect('/users');
26     }
27 });
```

Adding Authentication Routing (Cont'd)

```
29 /* Process the Login Request */
30 router.post('/login', passport.authenticate('local-login', {
31     successRedirect: '/users',
32     failureRedirect: '/login',
33     failureFlash: true
34 }));
35
36 /* Show Registration Page */
37 router.get('/register', function (req, res, next) {
38     if (!req.user) {
39         res.render('register', {
40             title: 'Register',
41             messages: req.flash('registerMessage'),
42             displayName: req.user ? req.user.displayName : ''
43         });
44     }
45     else {
46         return res.redirect('/');
47     }
48 });
```

Adding Authentication Routing (Cont'd)

```
50 /* POST signup data. */
51 router.post('/register', passport.authenticate('local-registration', {
52     //Success go to Profile Page / Fail go to Signup page
53     successRedirect : '/users',
54     failureRedirect : '/register',
55     failureFlash : true
56 }));
57
58
59 /* Process Logout Request */
60 router.get('/logout', function (req, res){
61   req.logout();
62   res.redirect('/');
63 });
64
65 module.exports = router;
```

Displaying flash error messages

- The **Connect-Flash** module is a node module that allows you to store temporary messages in an area of the session object called **flash**.
- Messages stored on the flash object will be cleared once they are presented to the user.
- This architecture makes the Connect-Flash module perfect to transfer messages before redirecting the request to another page.
  - `npm install connect-flash --save`

Configuring Connect-Flash module

- To configure your Express application to use the new **Connect-Flash** module, you'll have to require the new module in your Express configuration file and use the **`app.use()`** method to register it with your Express application.
  - `var flash = require('connect-flash');`

- After your view engine declaration:
  - `app.use(flash());`

Using Connect-Flash module

- Once installed, the **Connect-Flash** module exposes the `req.flash()` method, which allows you to create and retrieve **flash messages.**

```
15 /* Render Login page. */
16 router.get('/login', function (req, res, next) {
17     if (!req.user) {
18         res.render('login', {
19             title: 'Login',
20             messages: req.flash('loginMessage'),
21             displayName: req.user ? req.user.displayName : ''
22         });
23     }
24     else {
25         return res.redirect('/users');
26     }
27 });
28
29 /* Process the Login Request */
30 router.post('/login', passport.authenticate('local-login', {
31     successRedirect: '/users',
32     failureRedirect: '/login',
33     failureFlash: true
34 }));
```

Understanding Passport **Oauth** strategies

- **OAuth** is an authentication protocol that allows users to register with your web application using an **external provider**, without the need to input their username and password.
- OAuth is mainly used by social platforms, such as **Facebook**, **Twitter**, and **Google**, to allow users to register with other websites using their social account.

Setting up OAuth strategies

- **Passport** supports the basic **OAuth strategy**, which enables you to implement any **OAuth-based authentication**.
- However, it also supports a user authentication through major OAuth providers using wrapper strategies that help you avoid the need to implement a complex mechanism by yourself.

Example OAuth Strategy config

```
113     // Configure GitHub Strategy
114     passport.use(new GitHubStrategy({
115         clientID: GITHUB_CLIENT_ID,
116         clientSecret: GITHUB_CLIENT_SECRET,
117         callbackURL: "/auth/github/callback" // heroku deployment
118         //callbackURL: "http://127.0.0.1:3000/auth/github/callback" - before heroku deployment
119       },
120       function(accessToken, refreshToken, profile, done) {
121         User.findOne({ 'github.oauthID': profile.id }, function(err, user) {
122          if(err) { console.log(err); }
123          if (!err && user != null) {
124             done(null, user);
125          } else {
126
127              // create the user
128             var newUser = new User();
129             newUser.github.oauthID = profile.id;
130             newUser.github.name = profile.displayName;
131             newUser.github.created = Date.now();
132             newUser.save(function(err) {
133                 if (err) { throw err; }
134                 return done(null, newUser);
135             });
136
137          } // end-else
138         });
139     }));
```

Example Oauth Strategy routing

```
66  /* GITHUB Login Strategy Routes */
67  router.get('/auth/github', passport.authenticate('github'),
68      function(req, res){
69  });
70
71  router.get('/auth/github/callback', passport.authenticate('github', {
72      successRedirect : '/',
73      failureRedirect: '/login',
74      failureFlash:true
75  }));
76
```