

DECEMBER 09 2015

MEAN Stack User Registration and Login Example & Tutorial

After getting a lot of interest in a previous tutorial I wrote on user registration and login using AngularJS (<http://jasonwatmore.com/post/2015/03/10/AngularJS-User-Registration-and-Login-Example.aspx>), I thought I'd expand on it a bit further to show a full end to end solution including the server side and database using the MEAN stack (which stands for MongoDB, ExpressJS, AngularJS, NodeJS).

I find one of the best ways to learn how to use a new technology such as the MEAN stack is to tinker with and extend working examples of real world applications. User login and registration, security and account management are extremely common requirements for many applications.

The example project is available on GitHub at <https://github.com/cornflourblue/mean-stack-registration-login-example> (<https://github.com/cornflourblue/mean-stack-registration-login-example>)

Demo of MEAN Stack Tutorial Application

See a working demo of the MEAN Stack Tutorial application on runnable at http://code.runnable.com/Vmi69CdKg_Eacn_d/mean-stack-user-registration-and-login-example-for-node-js-angular-js-and-authentication (http://code.runnable.com/Vmi69CdKg_Eacn_d/mean-stack-user-registration-and-login-example-for-node-js-angular-js-and-authentication).

Runnable allows you to edit and run the application code in the browser, so you can learn how it all works without needing to install anything on your local machine.

MEAN Stack Dependencies

To run MEAN stack applications locally you need **NodeJS** installed and **MongoDB** running, for a guide on how to do this see [Setup the MEAN Stack on Windows](http://jasonwatmore.com/post/2015/12/26/Setup-the-MEAN-Stack-on-Windows.aspx) (<http://jasonwatmore.com/post/2015/12/26/Setup-the-MEAN-Stack-on-Windows.aspx>).

Project Setup

Once you've downloaded the code you can run the command '**npm install**' from the project root folder (where the package.json file is located) to download all node package dependencies.

Then run '**node server.js**' from the same location to start the web server and browse to <http://localhost:3000> to access the application.

MEAN Stack Tutorial Project Structure

I've split the login and registration pages out from the angular application in order to secure access to the angular client files, so all front end angular files (including javascript, css, images etc) are only available to authenticated users.

It's also possible to include the login and registration pages within the angular application and allow public access to the angular client files, this is how I've built previous examples and is fine as long as you don't have any sensitive data stored in your client files. The main benefit of securing the client files is just a bit of extra piece of mind that you're not going to accidentally leak any secure information in your angular app files.

The express js application is structured using an MVC-ish pattern, there are controllers and views but rather than models I've gone with a services layer for data access & business logic. The services use mongoskin as the mongodb driver, it's a thin wrapper for the native mongodb driver that provides a simpler interface for performing CRUD operations.

UPDATE 18 Apr 2016 - Replaced monk with mongoskin in order to update to the latest mongodb driver, monk still depends on an old version of the mongodb driver which was causing issues for some people.

Here's what the project structure looks like, click on any of the files or folders for the description and code:

- app
 - account
 - index.controller.js
 - index.html
 - app-content
 - app.css
 - app-services
 - flash.service.js
 - user.service.js
 - home
 - index.controller.js
 - index.html
 - app.js
 - index.html
- controllers
 - api
 - users.controller.js
 - app.controller.js
 - login.controller.js
 - register.controller.js
- services
 - user.service.js
- views
 - partials
 - footer.ejs
 - header.ejs
 - login.ejs
 - register.ejs
- config.json
- server.js

Angular App Folder

Path: /app

Back to top

The app folder contains all of the Angular application files, the angular project and code structure is largely based on recommendations from John Papa's angular style guide (<https://github.com/JohnPapa/angular-styleguide>), along with a few of my own tweaks.

Controllers and views are grouped by function/section (e.g. account, home), code that doesn't belong to a section is placed in a folder with an 'app-' prefix to prevent any name collisions and to make it easy to see what's what. For example css and images are located in the 'app-content' folder and angular services are located in the 'app-services' folder.

Angular Account Folder

Path: /app/account

Back to top

The account folder contains all controllers and views related to the account section of the angular application

Angular Account Index Controller

Path: /app/account/index.controller.js

[Back to top](#)

The account index controller is the default controller for the account section of the angular app, it makes the user object available to the view and exposes methods for updating or deleting the current user's account.

```
1  (function () {  
2    'use strict';  
3  
4    angular  
5      .module('app')  
6      .controller('Account.IndexController', Controller);  
7  
8    function Controller($window, UserService, FlashService) {  
9      var vm = this;  
10  
11      vm.user = null;  
12      vm.saveUser = saveUser;  
13      vm.deleteUser = deleteUser;  
14  
15      initController();  
16  
17      function initController() {  
18        // get current user  
19        UserService.GetCurrent().then(function (user) {  
20          vm.user = user;  
21        });  
22      }  
23  
24      function saveUser() {  
25        UserService.Update(vm.user)  
26          .then(function () {  
27            FlashService.Success('User updated');  
28          })  
29          .catch(function (error) {  
30            FlashService.Error(error);  
31          });  
32      }  
33  
34      function deleteUser() {  
35        UserService.Delete(vm.user._id)  
36          .then(function () {  
37            // log user out  
38            $window.location = '/login';  
39          })  
40          .catch(function (error) {  
41            FlashService.Error(error);  
42          });  
43      }  
44    }  
45  })();  
46
```

Angular Account Index View

Path: /app/account/index.html

[Back to top](#)

The account index view is the default view for the account section of the angular app, it contains a form for updating or deleting the current user's account.

```

1 <h1>My Account</h1>
2 <div class="form-container">
3   <form method="post">
4     <div class="form-group">
5       <label for="firstName">First name</label>
6       <input type="text" id="firstName" class="form-control" ng-model="vm.user.firstName" required />
7     </div>
8     <div class="form-group">
9       <label for="lastName">Last name</label>
10      <input type="text" id="lastName" class="form-control" ng-model="vm.user.lastName" required />
11    </div>
12    <div class="form-group">
13      <label for="username">Username</label>
14      <input type="text" id="username" class="form-control" ng-model="vm.user.username" required />
15    </div>
16    <div class="form-group">
17      <label for="password">Password</label>
18      <input type="password" id="password" class="form-control" ng-model="vm.user.password" />
19    </div>
20    <div class="form-group">
21      <button class="btn btn-primary" ng-click="vm.saveUser()">Save</button>
22      <button class="btn btn-danger" ng-click="vm.deleteUser()">Delete</button>
23    </div>
24  </form>
25 </div>

```

Angular App Content Folder

Path: /app/app-content

[Back to top](#)

The app content folder is used for static content such as css, images, fonts etc.

Angular App CSS

Path: /app/app-content/app.css

[Back to top](#)

The app css file is a stylesheet containing any custom styles for the angular application.

```

1 body {
2   padding: 20px 0;
3 }
4
5 .flash-message {
6   margin-top: 20px;
7 }

```

Angular Services Folder

Path: /app/app-services

[Back to top](#)

The services folder is used for custom AngularJS services / factories. All API access and business logic should be placed in services within this folder in order to keep angular controllers 'thin' and to maintain a clean separation of concerns.

Angular Flash Message Service

Path: /app/app-services/flash.service.js

[Back to top](#)

The flash message service is used to display success and error messages in the angular application. It uses the \$rootScope to expose the flash message to the main index.html file (/app/index.html) where the html is located for displaying the flash message.

The flash message is cleared on location change so it only displays once, optionally it can be kept after a single location change, this is if you want to display a flash message after redirecting the user.

```

1  (function () {
2    'use strict';
3
4    angular
5      .module('app')
6      .factory('FlashService', Service);
7
8    function Service($rootScope) {
9      var service = {};
10
11      service.Success = Success;
12      service.Error = Error;
13
14      initService();
15
16      return service;
17
18      function initService() {
19        $rootScope.$on('$locationChangeStart', function () {
20          clearFlashMessage();
21        });
22
23        function clearFlashMessage() {
24          var flash = $rootScope.flash;
25          if (flash) {
26            if (!flash.keepAfterLocationChange) {
27              delete $rootScope.flash;
28            } else {
29              // only keep for a single location change
30              flash.keepAfterLocationChange = false;
31            }
32          }
33        }
34      }
35
36      function Success(message, keepAfterLocationChange) {
37        $rootScope.flash = {
38          message: message,
39          type: 'success',
40          keepAfterLocationChange: keepAfterLocationChange
41        };
42      }
43
44      function Error(message, keepAfterLocationChange) {
45        $rootScope.flash = {
46          message: message,
47          type: 'danger',
48          keepAfterLocationChange: keepAfterLocationChange
49        };
50      }
51    }
52  })();
53
```

Angular User Service

Path: /app/app-services/user.service.js

[Back to top](#)

The user service encapsulates interaction with the web api for all user related CRUD operations.

```

1  (function () {
2    'use strict';
3
4    angular
5      .module('app')
6      .factory('UserService', Service);
7
8    function Service($http, $q) {
9      var service = {};
10
11      service.GetCurrent = GetCurrent;
12      service.GetAll = GetAll;
13      service.GetById = GetById;
14      service.GetByUsername = GetByUsername;
15      service.Create = Create;
16      service.Update = Update;
17      service.Delete = Delete;
18
19      return service;
20
21      function GetCurrent() {
22        return $http.get('/api/users/current').then(handleSuccess, handleError);
23      }
24
25      function GetAll() {
26        return $http.get('/api/users').then(handleSuccess, handleError);
27      }
28
29      function GetById(_id) {
30        return $http.get('/api/users/' + _id).then(handleSuccess, handleError);
31      }
32
33      function GetByUsername(username) {
34        return $http.get('/api/users/' + username).then(handleSuccess, handleError);
35      }
36
37      function Create(user) {
38        return $http.post('/api/users', user).then(handleSuccess, handleError);
39      }
40
41      function Update(user) {
42        return $http.put('/api/users/' + user._id, user).then(handleSuccess, handleError);
43      }
44
45      function Delete(_id) {
46        return $http.delete('/api/users/' + _id).then(handleSuccess, handleError);
47      }
48
49      // private functions
50
51      function handleSuccess(res) {
52        return res.data;
53      }
54
55      function handleError(res) {
56        return $q.reject(res.data);
57      }
58    }
59  })();
60

```

Angular Home Folder

Path: /app/home

[Back to top](#)

The home folder contains all controllers and views for the home section of the angular app.

Angular Home Index Controller

Path: /app/home/index.controller.js

[Back to top](#)

The home index controller is the default controller for the home section, it gets the current user and makes it available to the view.

```
1  (function () {  
2      'use strict';  
3  
4      angular  
5          .module('app')  
6          .controller('Home.IndexController', Controller);  
7  
8      function Controller(UserService) {  
9          var vm = this;  
10  
11          vm.user = null;  
12  
13          initController();  
14  
15          function initController() {  
16              // get current user  
17              UserService.GetCurrent().then(function (user) {  
18                  vm.user = user;  
19              });  
20          }  
21      }  
22  
23  })();
```

Angular Home Index View

Path: /app/home/index.html

[Back to top](#)

The home index view is the default view for the home section, it just displays a welcome message to the user.

```
1  <h1>Hi {{vm.user.firstName}}!!</h1>
```

Angular App.js File

Path: /app/app.js

[Back to top](#)

The app.js file is the entry point for the angular application where the app module is declared along with dependencies, and contains configuration and startup logic for when the app is first loaded.

The config function is used to define the routes of the application using the Angular UI Router.

The run function adds the JWT token as the default authorization header for all http requests made by the application, this is required to authenticate to the web api.

The last function in the file is used for manually bootstrapping the angular application after the JWT token is retrieved from the server, this is done to prevent the angular app from starting before the token is available.

```
1 (function () {  
2   'use strict';  
3  
4   angular  
5     .module('app', ['ui.router'])  
6     .config(config)  
7     .run(run);  
8  
9   function config($stateProvider, $urlRouterProvider) {  
10    // default route  
11    $urlRouterProvider.otherwise("/");  
12  
13    $stateProvider  
14      .state('home', {  
15        url: '/',  
16        templateUrl: 'home/index.html',  
17        controller: 'Home.IndexController',  
18        controllerAs: 'vm',  
19        data: { activeTab: 'home' }  
20      })  
21      .state('account', {  
22        url: '/account',  
23        templateUrl: 'account/index.html',  
24        controller: 'Account.IndexController',  
25        controllerAs: 'vm',  
26        data: { activeTab: 'account' }  
27      });  
28  }  
29  
30  function run($http, $rootScope, $window) {  
31    // add JWT token as default auth header  
32    $http.defaults.headers.common['Authorization'] = 'Bearer ' + $window.jwtToken;  
33  
34    // update active tab on state change  
35    $rootScope.$on('$stateChangeSuccess', function (event, toState, toParams, fromState, fromParams) {  
36      $rootScope.activeTab = toState.data.activeTab;  
37    });  
38  }  
39  
40  // manually bootstrap angular after the JWT token is retrieved from the server  
41  $(function () {  
42    // get JWT token from server  
43    $.get('/app/token', function (token) {  
44      window.jwtToken = token;  
45  
46      angular.bootstrap(document, ['app']);  
47    });  
48  });  
49 })();
```

Angular Index.html File

Path: /app/index.html

[Back to top](#)

The index.html file is the main html file for the angular application, it contains the overall template for the application.


```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <title>MEAN Stack User Registration and Login Example Application</title>
6
7   <link href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css" rel="stylesheet" />
8   <link href="app-content/app.css" rel="stylesheet" />
9 </head>
10 <body class="container">
11   <!-- header -->
12   <header>
13     <ul class="nav nav-tabs">
14       <li ng-class="{active: activeTab === 'home'}"><a ui-sref="home">Home</a></li>
15       <li ng-class="{active: activeTab === 'account'}"><a ui-sref="account">Account</a></li>
16       <li><a href="/login" target="_self">Logout</a></li>
17     </ul>
18     <div class="flash-message" ng-if="flash">
19       <div class="{alert alert-' + flash.type}" ng-bind="flash.message"></div>
20     </div>
21   </header>
22
23   <!-- main -->
24   <main ui-view></main>
25
26   <!-- footer -->
27   <footer></footer>
28
29   <!-- external scripts -->
30   <script src="//ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
31   <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.4.7/angular.min.js"></script>
32   <script src="//cdnjs.cloudflare.com/ajax/libs/angular-ui-router/0.2.15/angular-ui-router.min.js"></script>
33
34   <!-- application scripts -->
35   <script src="app.js"></script>
36   <script src="app-services/user.service.js"></script>
37   <script src="app-services/flash.service.js"></script>
38   <script src="home/index.controller.js"></script>
39   <script src="account/index.controller.js"></script>
40 </body>
41 </html>

```

Express/NodeJS Controllers Folder

Path: /controllers

[Back to top](#)

The express controllers folder contains all server-side controllers for the nodejs application.

Express/NodeJS API Controllers Folder

Path: /controllers/api

[Back to top](#)

The express api controllers folder contains all server-side controllers for the web api.

Express/NodeJS Users API Controller

Path: /controllers/api/users.controller.js

[Back to top](#)

The express users api controller defines the routes responsible for user related operations such as authentication, registration, retrieving, updating and deleting user data.

?

```
1 var config = require('config.json');
2 var express = require('express');
3 var router = express.Router();
4 var userService = require('services/user.service');
5
6 // routes
7 router.post('/authenticate', authenticateUser);
8 router.post('/register', registerUser);
9 router.get('/current', getCurrentUser);
10 router.put('/:_id', updateUser);
11 router.delete('/:_id', deleteUser);
12
13 module.exports = router;
14
15 function authenticateUser(req, res) {
16   userService.authenticate(req.body.username, req.body.password)
17     .then(function (token) {
18       if (token) {
19         // authentication successful
20         res.send({ token: token });
21       } else {
22         // authentication failed
23         res.sendStatus(401);
24       }
25     })
26     .catch(function (err) {
27       res.status(400).send(err);
28     });
29 }
30
31 function registerUser(req, res) {
32   userService.create(req.body)
33     .then(function () {
34       res.sendStatus(200);
35     })
36     .catch(function (err) {
37       res.status(400).send(err);
38     });
39 }
40
41 function getCurrentUser(req, res) {
42   userService.getById(req.user.sub)
43     .then(function (user) {
44       if (user) {
45         res.send(user);
46       } else {
47         res.sendStatus(404);
48       }
49     })
50     .catch(function (err) {
51       res.status(400).send(err);
52     });
53 }
54
55 function updateUser(req, res) {
56   var userId = req.user.sub;
57   if (req.params._id !== userId) {
58     // can only update own account
59     return res.status(401).send('You can only update your own account');
60   }
61
62   userService.update(userId, req.body)
63     .then(function () {
64       res.sendStatus(200);
65     })
66     .catch(function (err) {
67       res.status(400).send(err);
68     });
69 }
70
71 function deleteUser(req, res) {
72   var userId = req.user.sub;
73   if (req.params._id !== userId) {
74     // can only delete own account
75     return res.status(401).send('You can only delete your own account');
76   }
77
78   userService.delete(userId)
79     .then(function () {
80       res.sendStatus(200);
81     })
82     .catch(function (err) {
83       res.status(400).send(err);
84     });
85 }
```

Express/NodeJS App Controller

Path: /controllers/app.controller.js

[Back to top](#)

The express app controller controls access to the angular app client files. It uses session/cookie authentication to secure the angular files, and also exposes a JWT token to be used by the angular app to make authenticated api requests.

```
1 var express = require('express');
2 var router = express.Router();
3
4 // use session auth to secure the angular app files
5 router.use('/', function (req, res, next) {
6   if (req.path !== '/login' && !req.session.token) {
7     return res.redirect('/login?returnUrl=' + encodeURIComponent('/app' + req.path));
8   }
9
10  next();
11 });
12
13 // make JWT token available to angular app
14 router.get('/token', function (req, res) {
15   res.send(req.session.token);
16 });
17
18 // serve angular app files from the '/app' route
19 router.use('/', express.static('app'));
20
21 module.exports = router;
```

Express/NodeJS Login Controller

Path: /controllers/login.controller.js

[Back to top](#)

The express login controller defines routes for displaying the login view and authenticating user credentials. It uses the api to authenticate rather than using the user service directly, this is to keep the api and database layers cleanly separated from the rest of the application so they could easily be split if necessary and run on separate servers.

On successful authentication the jwt token returned from the api is stored in the user session so it can be made available to the angular application when it loads (via the '/token' route defined in the express app controller above).

```

1  var express = require('express');
2  var router = express.Router();
3  var request = require('request');
4  var config = require('config.json');
5
6  router.get('/', function (req, res) {
7    // log user out
8    delete req.session.token;
9
10   // move success message into local variable so it only appears once (single read)
11   var viewData = { success: req.session.success };
12   delete req.session.success;
13
14   res.render('login', viewData);
15 });
16
17 router.post('/', function (req, res) {
18   // authenticate using api to maintain clean separation between layers
19   request.post({
20     url: config.apiUrl + '/users/authenticate',
21     form: req.body,
22     json: true
23   }, function (error, response, body) {
24     if (error) {
25       return res.render('login', { error: 'An error occurred' });
26     }
27
28     if (!body.token) {
29       return res.render('login', { error: 'Username or password is incorrect', username: req.body.username });
30     }
31
32     // save JWT token in the session to make it available to the angular app
33     req.session.token = body.token;
34
35     // redirect to returnUrl
36     var returnUrl = req.query.returnUrl && decodeURIComponent(req.query.returnUrl) || '/';
37     res.redirect(returnUrl);
38   });
39 });
40
41 module.exports = router;

```

Express/NodeJS Register Controller

Path: /controllers/register.controller.js

[Back to top](#)

The express register controller defines routes for displaying the register view and registering a new user. As with the login controller (above) it uses the api to register new users in order to maintain clean separation from the api and database layers.

```

1  var express = require('express');
2  var router = express.Router();
3  var request = require('request');
4  var config = require('config.json');
5
6  router.get('/', function (req, res) {
7    res.render('register');
8  });
9
10 router.post('/', function (req, res) {
11   // register using api to maintain clean separation between layers
12   request.post({
13     url: config.apiUrl + '/users/register',
14     form: req.body,
15     json: true
16   }, function (error, response, body) {
17     if (error) {
18       return res.render('register', { error: 'An error occurred' });
19     }
20
21     if (response.statusCode !== 200) {
22       return res.render('register', {
23         error: response.body,
24         firstName: req.body.firstName,
25         lastName: req.body.lastName,
26         username: req.body.username
27       });
28     }
29
30     // return to login page with success message
31     req.session.success = 'Registration successful';
32     return res.redirect('/login');
33   });
34 });
35
36 module.exports = router;

```

Express/NodeJS Services Folder

Path: /services

[Back to top](#)

The express services folder is for the services layer of the node application, where all business logic and data access is located.

Express/NodeJS User Service

Path: /services/user.service.js

[Back to top](#)

The express user service encapsulates all data access and business logic for users behind a simple interface. It exposes methods for CRUD operations and user authentication.

I've implemented all of the service methods using promises in order to keep the users api controller simple and consistent, so all service methods can be called with the pattern `[service method].then(...).catch(...);`

Mongoskin is the MongoDB driver used to access to the database, it provides a thin layer over the native mongodb driver that makes it a bit simpler to perform CRUD operations.

```

1  var config = require('config.json');
2  var _ = require('lodash');
3  var jwt = require('jsonwebtoken');
4  var bcrypt = require('bcryptjs');
5  var Q = require('q');
6  var mongo = require('mongoskin');

```

```
7 var db = mongo.db(config.connectionString, { native_parser: true });
8 db.bind('users');
9
10 var service = {};
11
12 service.authenticate = authenticate;
13 service.getById = getById;
14 service.create = create;
15 service.update = update;
16 service.delete = _delete;
17
18 module.exports = service;
19
20 function authenticate(username, password) {
21   var deferred = Q.defer();
22
23   db.users.findOne({ username: username }, function (err, user) {
24     if (err) deferred.reject(err);
25
26     if (user && bcrypt.compareSync(password, user.hash)) {
27       // authentication successful
28       deferred.resolve(jwt.sign({ sub: user._id }, config.secret));
29     } else {
30       // authentication failed
31       deferred.resolve();
32     }
33   });
34
35   return deferred.promise;
36 }
37
38 function getById(_id) {
39   var deferred = Q.defer();
40
41   db.users.findById(_id, function (err, user) {
42     if (err) deferred.reject(err);
43
44     if (user) {
45       // return user (without hashed password)
46       deferred.resolve(_.omit(user, 'hash'));
47     } else {
48       // user not found
49       deferred.resolve();
50     }
51   });
52
53   return deferred.promise;
54 }
55
56 function create(userParam) {
57   var deferred = Q.defer();
58
59   // validation
60   db.users.findOne(
61     { username: userParam.username },
62     function (err, user) {
63       if (err) deferred.reject(err);
64
65       if (user) {
66         // username already exists
67         deferred.reject('Username "' + userParam.username + '" is already taken');
68       } else {
69         createUser();
70       }
71     }
72   );
73
74   function createUser() {
75     // set user object to userParam without the cleartext password
76     var user = _.omit(userParam, 'password');
77
78     // add hashed password to user object
79     user.hash = bcrypt.hashSync(userParam.password, 10);
80
81     db.users.insert(
82       user,
83       function (err, doc) {
84         if (err) deferred.reject(err);
85
86         deferred.resolve();
87       }
88     );
89
90     return deferred.promise;
91   }
92 }
93
94 function update(_id, userParam) {
95   var deferred = Q.defer();
96
97   // validation
98   db.users.findById(_id, function (err, user) {
```

```

97         if (err) deferred.reject(err);
98
99         if (user.username !== userParam.username) {
100             // username has changed so check if the new username is already taken
101             db.users.findOne(
102                 { username: userParam.username },
103                 function (err, user) {
104                     if (err) deferred.reject(err);
105
106                     if (user) {
107                         // username already exists
108                         deferred.reject('Username "' + req.body.username + '" is already taken')
109                     } else {
110                         updateUser();
111                     }
112                 });
113             } else {
114                 updateUser();
115             }
116         });
117
118         function updateUser() {
119             // fields to update
120             var set = {
121                 firstName: userParam.firstName,
122                 lastName: userParam.lastName,
123                 username: userParam.username,
124             };
125
126             // update password if it was entered
127             if (userParam.password) {
128                 set.hash = bcrypt.hashSync(userParam.password, 10);
129             }
130
131             db.users.update(
132                 { _id: mongo.helper.toObjectID(_id) },
133                 { $set: set },
134                 function (err, doc) {
135                     if (err) deferred.reject(err);
136
137                     deferred.resolve();
138                 });
139         }
140
141         return deferred.promise;
142     }
143
144     // prefixed function name with underscore because 'delete' is a reserved word in javascript
145     function _delete(_id) {
146         var deferred = Q.defer();
147
148         db.users.remove(
149             { _id: mongo.helper.toObjectID(_id) },
150             function (err) {
151                 if (err) deferred.reject(err);
152
153                 deferred.resolve();
154             });
155
156         return deferred.promise;
157     }

```

Express/NodeJS Views Folder

Path: /views

Back to top

The express views folder contains all views for the nodejs application, the EJS (embedded javascript) view engine is being used.

Express/NodeJS Partial Views Folder

Path: /views/partials

[Back to top](#)

The express partial views folder contains all partial views for the nodejs application such as the header and footer. The EJS view engine doesn't support layouts which is why I went with partials instead for the header and footer.

Express/NodeJS Footer Partial View

Path: /views/partials/footer.ejs

[Back to top](#)

The footer partial contains the html for the bottom of the page layout, which in this case is simply the closing tags for body and html.

```
1 </body>
2 </html>
```

Express/NodeJS Header Partial View

Path: /views/partials/header.ejs

[Back to top](#)

The header partial contains the html for the top of the page layout.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <title>Login</title>
6   <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css" />
7   <style type="text/css">
8     body {
9       padding-top: 100px;
10    }
11
12    .form-container {
13      width: 400px;
14      margin: auto;
15    }
16  </style>
17 </head>
18 <body>
```

Express/NodeJS Login View

Path: /views/login.ejs

[Back to top](#)

The login view contains a simple form for signing into the application.

```

1 <%- include('partials/header') %>
2 <div class="form-container">
3   <h2>Login</h2>
4   <% if(locals.error) { %>
5     <div class="alert alert-danger"><%= error %></div>
6   <% } %>
7   <% if(locals.success) { %>
8     <div class="alert alert-success"><%= success %></div>
9   <% } %>
10  <form method="post">
11    <div class="form-group">
12      <label for="username">Username</label>
13      <input type="text" name="username" id="username" value="<%= locals.username || '' %>" class="form-control" />
14    </div>
15    <div class="form-group">
16      <label for="password">Password</label>
17      <input type="password" name="password" id="password" class="form-control" />
18    </div>
19    <div class="form-group">
20      <button type="submit" class="btn btn-primary">Login</button>
21      <a href="/register" class="btn btn-link">Register</a>
22    </div>
23  </form>
24 </div>
25 <%- include('partials/footer') %>

```

Express/NodeJS Register View

Path: /views/register.ejs

[Back to top](#)

The register view contains a simple form for registering an account with the application.

```

1 <%- include('partials/header') %>
2 <div class="form-container">
3   <h2>Register</h2>
4   <% if(locals.error) { %>
5     <div class="alert alert-danger"><%= error %></div>
6   <% } %>
7   <form method="post">
8     <div class="form-group">
9       <label for="firstName">First name</label>
10      <input type="text" name="firstName" id="firstName" class="form-control" value="<%= locals.firstName || '' %>" required />
11    </div>
12    <div class="form-group">
13      <label for="lastName">Last name</label>
14      <input type="text" name="lastName" id="lastName" class="form-control" value="<%= locals.lastName || '' %>" required />
15    </div>
16    <div class="form-group">
17      <label for="username">Username</label>
18      <input type="text" name="username" id="username" class="form-control" value="<%= locals.username || '' %>" required />
19    </div>
20    <div class="form-group">
21      <label for="password">Password</label>
22      <input type="password" name="password" id="password" class="form-control" required />
23    </div>
24    <div class="form-group">
25      <button type="submit" class="btn btn-primary">Register</button>
26      <a href="/login" class="btn btn-link">Cancel</a>
27    </div>
28  </form>
29 </div>
30 <%- include('partials/footer') %>

```

Express/NodeJS App Config File

Path: /config.json

[Back to top](#)

The express config file contains configuration data used by the nodejs application.

```
1 {
2   "connectionString": "mongodb://localhost:27017/mean-stack-registration-login-example (mongodb://localhost:27017/mean-stack-registration-
3   "apiUrl": "http://localhost:3000/api (http://localhost:3000/api)",
4   "secret": "REPLACE THIS WITH YOUR OWN SECRET, IT CAN BE ANY STRING"
5 }
```

Express/NodeJS Server File

Path: /server.js

[Back to top](#)

The express server file is the entry point into the nodejs application, it defines app wide settings, binds controllers to routes and starts the http server.

```
1 require('rootpath')();
2 var express = require('express');
3 var app = express();
4 var session = require('express-session');
5 var bodyParser = require('body-parser');
6 var expressJwt = require('express-jwt');
7 var config = require('config.json');
8
9 app.set('view engine', 'ejs');
10 app.set('views', __dirname + '/views');
11 app.use(bodyParser.urlencoded({ extended: false }));
12 app.use(bodyParser.json());
13 app.use(session({ secret: config.secret, resave: false, saveUninitialized: true }));
14
15 // use JWT auth to secure the api
16 app.use('/api', expressJwt({ secret: config.secret }).unless({ path: ['/api/users/authenticate', '/api/users/register'] }));
17
18 // routes
19 app.use('/login', require('./controllers/login.controller'));
20 app.use('/register', require('./controllers/register.controller'));
21 app.use('/app', require('./controllers/app.controller'));
22 app.use('/api/users', require('./controllers/api/users.controller'));
23
24 // make '/app' default route
25 app.get('/', function (req, res) {
26   return res.redirect('/app');
27 });
28
29 // start server
30 var server = app.listen(3000, function () {
31   console.log('Server listening at http:// (http://)' + server.address().address + ':' + server.address().port);
32 });
```

Related Posts

- [MEANie - Lightweight MEAN Stack Blogging Platform & CMS \(/post/2016/10/29/meanie-mean-stack-blogging-platform\)](#) - Oct 29 2016

MEAN Stack Web Development Sydney

Feel free to drop me a line (</contact.aspx>) if you're looking for web development or consulting services in Sydney Australia, I also provide remote contracting services for clients outside Sydney.




By Jason Watmore (https://twitter.com/jason_watmore)

Tags: [MEAN Stack \(/posts/tag/mean-stack\)](/posts/tag/mean-stack), [Login \(/posts/tag/login\)](/posts/tag/login), [Registration \(/posts/tag/registration\)](/posts/tag/registration), [Authentication and Authorization \(/posts/tag/authentication-and-authorization\)](/posts/tag/authentication-and-authorization)

Share on:

ABOUT

I'm a web developer in Sydney Australia and the owner of Point Blank Development (<https://www.pointblankdevelopment.com.au>), I've been building websites and web applications in Sydney since 1998.

Follow me on:  (<https://github.com/cornflourblue>)  (<http://stackoverflow.com/users/2955770/jason>)
 (https://twitter.com/jason_watmore)

MONTHS

2016

December (/posts/2016/12) (3)
November (/posts/2016/11) (2)
October (/posts/2016/10) (1)
September (/posts/2016/09) (1)
August (/posts/2016/08) (3)
July (/posts/2016/07) (3)
June (/posts/2016/06) (1)
May (/posts/2016/05) (1)
April (/posts/2016/04) (1)
March (/posts/2016/03) (1)
February (/posts/2016/02) (1)
January (/posts/2016/01) (2)

2015

December (/posts/2015/12) (2)
November (/posts/2015/11) (1)
October (/posts/2015/10) (1)
July (/posts/2015/07) (1)
June (/posts/2015/06) (1)
April (/posts/2015/04) (1)
March (/posts/2015/03) (2)
January (/posts/2015/01) (1)

2014

December (/posts/2014/12) (1)
September (/posts/2014/09) (1)
August (/posts/2014/08) (1)
July (/posts/2014/07) (1)
May (/posts/2014/05) (2)
April (/posts/2014/04) (3)
March (/posts/2014/03) (2)
February (/posts/2014/02) (2)

2013

November (/posts/2013/11) (1)
October (/posts/2013/10) (1)
July (/posts/2013/07) (1)
April (/posts/2013/04) (1)

2012

October (/posts/2012/10) (1)
September (/posts/2012/09) (1)
August (/posts/2012/08) (1)
July (/posts/2012/07) (1)
June (/posts/2012/06) (1)
May (/posts/2012/05) (1)
April (/posts/2012/04) (1)
March (/posts/2012/03) (1)
February (/posts/2012/02) (1)

2011

December (/posts/2011/12) (1)
November (/posts/2011/11) (3)

TAGS

ASP.NET (/posts/tag/aspnet)
ASP.NET Web API (/posts/tag/aspnet-web-api)
Angular 2 (/posts/tag/angular-2)
Angular Directive (/posts/tag/angular-directive)
Angular UI Router (/posts/tag/angular-ui-router)
AngularJS (/posts/tag/angularjs)
Authentication and Authorization (/posts/tag/authentication-and-authorization)
Bootstrap (/posts/tag/bootstrap)
C# (/posts/tag/c)
CSS3 (/posts/tag/css3)
Chai (/posts/tag/chai)
DDD (/posts/tag/ddd)
Design Patterns (/posts/tag/design-patterns)
Dynamic LINQ (/posts/tag/dynamic-linq)
ELMAH (/posts/tag/elmah)
Exchange (/posts/tag/exchange)
Facebook (/posts/tag/facebook)
Fluent NHibernate (/posts/tag/fluent-nhibernate)
Google API (/posts/tag/google-api)
Google Analytics (/posts/tag/google-analytics)
Google Maps API (/posts/tag/google-maps-api)
Google Plus (/posts/tag/google-plus)
HTML5 (/posts/tag/html5)
HTTP (/posts/tag/http)
Heroku (/posts/tag/heroku)
IIS (/posts/tag/iis)
Insecure Content (/posts/tag/insecure-content)
Instagram API (/posts/tag/instagram-api)