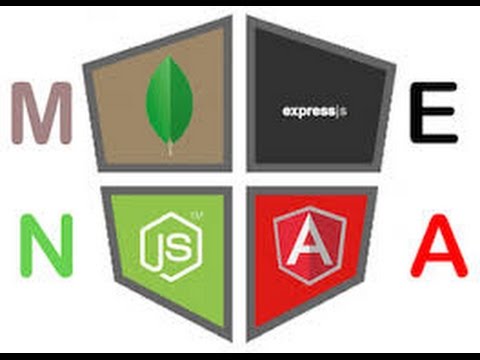
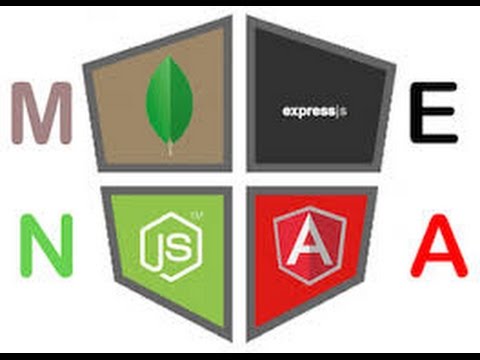
**Exercise06\_04\_01 – Step 1**

1. To set up Login, we can make use of Satellizer. We will start off with the configuration in index.config.js. We will set up a login URL as follows:  
    $authProvider.signupUrl = API\_URL + 'auth/register';  
    ***$authProvider.loginUrl = API\_URL + 'auth/login';***
2. Now we can work on the Login area of auth.html. We will start with an ***ng-submit*** directive for the Login form:  
    <div class="panel-heading">Login</div>  
    <div class="panel-body">  
    <form ***ng-submit="auth.login()"***>
3. Let’s set up a model for our Email ***<input>*** field:  
    <label>Email address</label>  
    <input type="email" class="form-control"  
    ***ng-model="auth.login.user.email"***>
4. Now let’s set up a model for our Password ***<input>*** field:  
    <label>Email address</label>  
    <input type="email" class="form-control"  
    ***ng-model="auth.login.user.pwd"***>
5. Now we can go to our ***auth.controller.js*** and create our ***login()*** function. Let’s copy the ***register()*** function and we can modify it as follows:  
    ***login***() {  
    var vm = this;  
    this.$auth.***login***(this.***login***.user)  
    .then(function(token) {  
    vm.$auth.setToken(token);  
    });  
    }  
   Let’s give a ***Login*** a try with a previously registered user. Our Developer Tools ***Network*** tab should give us a ***login*** request error of ***404 Not Found***. this is expected. It confirms that our message has been sent, but there is not yet an ***endpoint*** to receive it on the server.

**Exercise06\_04\_01 – Step 2**

1. Switch over to the ***back-end***; we will need a route to support ***Login***. Let’s set that up in ***server.js***, just below our ***/auth/register*** route:  
   app.post('/auth/register', auth.register);  
     
   ***app.post('/auth/login', auth.login);***
2. We can build the support function for the route ***callback*** in our ***auth.js*** controller. Let’s scaffold that out:  
    }***,  
    login : (req, res) => {  
      
    }***
3. We can use some similar code to our ***register()*** function. First we will trap that the user is not registered, as follows:  
    login : (req, res) => {  
    ***User.findOne({  
    email: req.body.email  
    }, (err, user) => {  
    if (!user) {  
    return res.status(401).send({  
    message: 'Email or Password invalid'  
    })  
    }  
    });*** }
4. If we find the user, we need to check that the ***password*** matches. If it does, we will create our ***token***, if not we will send an ***error***. Let’s do that as follows, with some debug code to help:  
    if (!user) {  
    return res.status(401).send({  
    message: 'Email or Password invalid'  
    })  
    }  
    ***if (req.body.pwd == user.pwd) {  
    console.log(req.body, user.pwd);  
    res.send({  
    token: createToken(user)  
    });  
    }  
    else {  
    return res.status(401),send({  
    message: 'Invalid email and/or password'  
    })  
    }***Now let’s run the server and do a test with a user that we have registered. First, let’s clear the Satellizer token from ***local*** ***storage***. When we login with an existing user, we should get a new token. Now let’s login as an unregistered user, which should produce our ***401 Unauthorized*** error. Try again with a real user, and it should be working.

**Exercise06\_04\_01 – Step 3**

1. Now let’s close things up by building Logout functionality. If the user is logged in, we want to switch the Login button on the Navbar to a Logout. Let’s switch over to the ***front-end*** and open up the ***/src/app/components/navbar*** folder. This area had no AngularJS functionality before, but now we are going to need a controller for it. Create a file in the folder called navbar.controller.js.
2. Let’s scaffold out a controller ***class*** called ***NavbarController*** to export. It will need a ***constructor***. That constructor will have to ***inject*** the ***$auth*** service so we can ascertain if the user is logged in. Let’s do that as follows:  
   ***export class NavbarController {  
    constructor($auth) {  
    'ngInject';  
    }  
   }***
3. Now let’s open up ***navbar.html***. We will need to add an **ng-controller** directive to it, because this partial is not in our ***ui-view***. We can use some syntax to ***nickname*** it as follows:  
   <nav class="navbar navbar-static-top navbar-inverse"  
    ***ng-controller="NavbarController as nav"***>
4. We will now need to register our new controller in the app. We will do that in ***index.module.js***, by first importing the code into the file below our AuthController:  
   import { AuthController } from './auth/auth.controller';  
   **import { NavbarController } from   
    './components/navbar/navbar.controller';**
5. We then need to register it, using a chained ***controller()*** method from our ***angular.module()*** statement:  
    .controller('AuthController', AuthController)  
    ***.controller('NavbarController', NavbarController)***
6. Let’s continue to build out our NavbarController in ***navbar.controller.js***. We can use an ***isAuthenticated*** method of the ***Satellizer*** service to determine the authentication state. We will set that method as a ***property*** of the controller in its ***constructor***:  
    constructor($auth) {  
    'ngInject';  
    ***this.isAuthenticated = $auth.isAuthenticated;***  
    }
7. In ***navbar.htm***l, we can use an ng-hide directive to hide the Login button if the controller tells us that the user is authenticated as follows:  
    <li ***ng-hide="nav.isAuthenticated()"***>  
    <a ui-sref="auth">Login</a></li>  
   Let’s do a test with a user that we have registered. First, let’s clear the Satellizer token from ***local*** ***storage***. Regenerate the auth page and the Login button will be there. Now login a registered user and the Login button should disappear.
8. Now let’s build a ***Logout*** button which we will use ***ng-show*** to display if the user is authenticated:  
    <li ng-hide="nav.isAuthenticated()">  
    <a ui-sref="auth">Login</a></li>  
    ***<li ng-show="nav.isAuthenticated()">  
    <a ui-sref="auth">Logout</a></li>***  
   Let’s try the same test and see if it is working.
9. Let’s create a ***logout()*** function in our ***navbar.controller.js***. It will use another Satellizer method to remove our token for us from ***local*** ***storage***. So we will need to get a ***reference*** to it in our constructor, then build ***logout()*** and use it:  
    constructor($auth) {  
    'ngInject';  
    ***this.$auth = $auth;*** this.isAuthenticated = $auth.isAuthenticated;  
    }  
      
    ***logout() {  
    this.$auth.logout();  
    }***
10. Let’s go back to ***navbar.html*** to implement the ***click*** on Logout. We can change the ***ui-sref*** directive to an ***ng-click*** directive to call our logout function as follows:  
     <li ng-show="nav.isAuthenticated()">  
     <a ***ng-click="nav.logout()"***>Logout</a></li>  
    To test this, make sure we are logged in and set the Developer tools to application so we can see the token. Click Logout, and the token should disappear.

We are now done with a message board, which contains an end-to-end authentication system.