Network Programming
Lecture 6: Registration

Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

Elgayyar.weebly.com

Registration

1. ***Create Registration View in Angular***
2. *Use a password match Angular directive*
3. *Setup Satellizer for registration*
4. *Create a registration endpoint with Express*
5. *Save a user with Mongo*
6. *Associate a user with post when making a post*

Change NavBar

```
        ..
              <span class="glyphicon glyphicon-home"></span>
              My Message Board
       </a>
     </div>
        …
       <ul class="nav navbar-nav">
         <li class="active"><a ng-href="#">Home</a></li>
         <li><a ng-href="#/auth">Login</a></li>
       </ul>
     </div>

   </div>
 </nav>
```

App/components/navbar/navbar.html

Creaet an Authentication View -1

- *Create a folder called **auth** in your app and create two files inside it*
  - auth.html (just add hello world inside it for testing)
  - auth.controller.js
- *Copy main controller class definition (first line) to auth one*
  - Rename the class to AuthController
- *Add new state in the router (index.router.js)*

```
$stateProvider
.state('home', {
   url: '/',
    templateUrl: 'app/main/main.html',
    controller: 'MainController',
    controllerAs: 'main'
    })                          //remove ;

.state('auth', {
   url: '/auth',
   templateUrl: 'app/auth/auth.html',
   controller: 'AuthController',
controllerAs: 'auth'});
```

App/auth, index.router.js

Creaet an Authentication View -2

- *To be able to use our new state we have to register inside index.module.js*

```
import { MainController } from './main/main.controller';
import { AuthController } from './auth/auth.controller‘;
 …
.controller('MainController', MainController)
.controller('AuthController', AuthController)
```

- *Create page layout*
  - See auth.html
  - Two panels (one for login and one for registration) each contains a form with a submit button
  - <col-md-6> tag for responsiveness, in small screens it will be over each other

index.module.js, auth.html

Registration

1. *Create Registration View in Angular*
2. ***Use a password match Angular directive***
3. *Setup Satellizer for registration*
4. *Create a registration endpoint with Express*
5. *Save a user with Mongo*
6. *Associate a user with post when making a post*

Use a Validation Directive

- *Use an angular validation match directive from here*
- *Install (while inside the front end folder)*

```
bower install angular-validation-match --save
```

- *Inject angular-validation-match into your module (index.module.js)*

```
angular.module('myApp',['validation.match'])
```

- *Update the form code as shown in the example*
- *Add the css styles to view the red lines around the boxes.*

Registration

1. *Create Registration View in Angular*
2. *Use a password match Angular directive*
3. ***Setup Satellizer for registration***
4. *Create a registration endpoint with Express*
5. *Save a user with Mongo*
6. *Associate a user with post when making a post*

Install Satellizer

- *Satellizer: end-to-end token-based authentication module for AngularJS*
  - Allow you to login using Facebook, Google, and other social accounts
- *Install (while inside the front end folder)*

```
bower install satellizer --save
```

- *Inject angular-validation-match into your module (index.module.js)*

```
angular.module('myApp',['satellizer'])
```

- *Configuration (index.config.js)*
  - Set URL with a constant in index.module.js (`.constant('API_URL','http://localhost:5000')`)

```
export function config ($logProvider, toastrConfig, $authProvider, API_URL) {
  'ngInject';
  …
  $authProvider.signupUrl = API_URL + 'auth/register';

}
```

Connect Satellizer to the form Submit

- *Add ng-submit to the register form (auth.html)*

```
<form name="reigster" ng-submit="auth.register()">
```

- *Create the register function (auth.controller.js)*
  - Test in your browser and hit submit in register form. You should get **404 error!!**

```
export class AuthController {

    constructor($auth){   //satellizer service
        'ngInject';

        this.$auth = $auth;
    }

    register() {
        this.$auth.signup({email: 'test@test.com'});
    }
}
```

Registration

1. *Create Registration View in Angular*
2. *Use a password match Angular directive*
3. *Setup Satellizer for registration*
4. ***Create a registration endpoint with Express***
5. *Save a user with Mongo*
6. *Associate a user with post when making a post*

Create a registration endpoint with Express

- *Add new endpoint (server.js)*

```
app.post('/auth/register', function(req,res){
        console.log(req.body);
})}
```

- *Re-run your server and test angular request.*

Pass the actual email and password to the registration API

- *Add ng-model (auth.html), we have it already for password (ensure it is started with auth.user.*) , so just add it on the email*

```
<div class="form-group">
        <label>Email address</label>
        <input type="email" class="form-control" ng-model="auth.user.email">
</div>
```

- *Send this.user to the auth API*

```
export class AuthController {

    constructor($auth){   //satellizer service
        'ngInject';

        this.$auth = $auth;
    }

    register() {
        this.$auth.signup(this.user);
    }
}
```

Disable Submit Button if Password Dis-match

- *Add ng-disabled to the submit button (auth.html)*

```
<button type="submit" class="btn btn-default“
         ng-disabled=“register.myConfirmField.$error.match”>
         Submit
</button>
```

Registration

1. *Create Registration View in Angular*
2. *Use a password match Angular directive*
3. *Setup Satellizer for registration*
4. *Create a registration endpoint with Express*
5. ***Save a user with Mongo***
6. *Associate a user with post when making a post*

Clear Up Your Backend

- *Move the message model to a separate file*
  - Create a folder called **models**
  - Create **Message.js** inside the folder
  - Cut and paste the message model form the server.js
  - Instead of saving it in a variable, just export it so it can be accessed outside

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

module.exports = mongoose.model('Message',{
    msg: String
});
```

  - Now you can import your new model inside your server

```
var Message= require ('./models/message');
```

  - Test your app to ensure no errors…

Create User Model

- Create a new file **user.js** inside models folder

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

module.exports = mongoose.model(‘User',{
    email: String,
    pwd: String
});
```

- Now you can import your new model inside your server

```
var User= require (‘./models/user’);
```

- No update your post message to use the new model

```
app.post('/auth/register', function(req,res){
        console.log(req.body);
        var user= new User(req.body);
        user.save(function(err,result){
                if(err)
                   res.status(500).send({message:err.message});

                res.status(200);

        });

})}
```

Test to Submit More than one Time

- Run your server, and test to submit the same email more than one time
- Check the results in the users collection in mongo console !!
- You will find the same email is stored several times and this is a not acceptable behavior
- So we have to fix that
- But first !!! Clear your server a little bit more ☺
- Move API functions inside new modules

Move API Functions to New Modules

- Create a new folder called controllers
- Create a file called **auth.js**
- Export an object that contains the required function

```
var User = require('../models/user');

module.exports = {
    register: function (req, res) {
        var user = new User(req.body);

         user.save(function (err, result) {
               if (err) {
                   res.status(500).send({
                       message: err.message
                   });
               }
               res.status(200);
           })
        });
    }
}
```

- Now use the object in your server.js

```
var auth = require('./controllers/auth');
app.post('/auth/register', auth.register);
```

Do the same for /api/message

Test for Email Duplication

```
module.exports = {
    register: function (req, res) {

        User.findOne({
            email: req.body.email
        }, function (err, existingUser) {

            if(existingUser)
                return res.status(409).send({message: 'Email is already registered'});

            var user = new User(req.body);

            user.save(function (err, result) {
                if (err) {
                    res.status(500).send({
                        message: err.message
                    });
                }
                res.status(200);
            })
        });
    }
}
```

Let us Continue our Authentication

- *We have to give the browser something back to authenticate the user.*
  - Token Authentication
  - We need a token library for our server → **JWT (JSON Web Token Library)**
  - Navigate to your backend folder and install JWT

```
npm install jwt-simple -save
```

  - Require in your auth controller (auth.js)

```
var jwt= require ('jwt-simple');
```

  - Create a function that generates the token under your module.exports

```
function createToken(user){ //should be part of the token

}
```

createToken Function

- You need to create a payload to generate a token:
  - User info (here just user id)
  - Creation time
  - Expiration time
  - To get time, we use another library called *moment (npm install moment --save)*

```
//add in the top
var moment = require(‘moment’);

function createToken(user){                     //should be part of the token
   var payload={
      sub: user._id,
      iat: moment( ).unix(),                    //issued at time (current time)
      exp: moment( ).add(14,’days’).unix()
   };
   return jwt.encode(payload, ‘secret’);   //encode against token secret
   //secret should be more complex and in config file
}
```

Send the Token Back to the Front-End

```
module.exports = {
    register: function (req, res) {

        User.findOne({
            email: req.body.email
        }, function (err, existingUser) {

            if(existingUser)
                return res.status(409).send({message: 'Email is already registered'});

            var user = new User(req.body);

            user.save(function (err, result) {
                if (err) {
                    res.status(500).send({
                        message: err.message
                    });
                }
                res.status(200).send ({token:createToken(result)});
            })
        });
    }
}
```

Store Your Token in Browser

- Restart Your Server
- Register a new User
- You should get back a token
  - Simply check the response of your request (inspect→ Network)
- We have to use satellizer to store this token in browser local storage
  - Change register method in **auth.controller.js**

```
register() {
     var vm=this;
     this.$auth.signup(this.user).then(function(token){
               vm.$auth.setToken(token);
      });
}
```

- Now, register a user and check your token in browser local storage
  - Inspect → Resources → local Storage (you should see satellizer_token)

Registration

1. *Create Registration View in Angular*
2. *Use a password match Angular directive*
3. *Setup Satellizer for registration*
4. *Create a registration endpoint with Express*
5. *Save a user with Mongo*
6. ***Associate a user with post when making a post***

User Authorization

- Satellizer attach an existing token to each request we send to our server in the authorization header
  - The token includes: user_id, creation time, and expiration time.
  - So our node app should get user id from the request and ensure that token is not expired.
- In your server.js file create a new **middleware function**

```
var jwt=require(‘jwt-simple’);
var moment= require(‘moment’);
function checkAuthenticated(req, res, next) {
       if(!req.header('Authorization')) {
                return res.status(401).send({message:
                'Please make sure your request has an Authorization header'});
       }
       var token = req.header('Authorization').split(' ')[1]; //JWT, Basic , Bearer
       var payload = jwt.decode(token, 'secret');
       if(payload.exp <= moment().unix()){
          return res.status(401).send({message: 'Token has expired'});
       }

       req.user = payload.sub;                                  //add it to the request
       next();
}
```

Ensure User Authentication on Post Message

- Ensure that your middleware function are involved during message post

```
app.post('/api/message',checkAuthenticated, message.post);
```

- Re-Run your server and give it a try

  1. Post a message from your front-end, you should see it on your server console
  2. Delete the token from your local storage
  3. Try to post a message, you should get 401 error, missing authorization header

Update your Message Model

- *Add new property to the message model (Message.js)*

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

module.exports = mongoose.model('Message',{
    msg: String,
    user: {type: mongoose.Schema.ObjectId, ref: 'User'}
});
```

Update Your Post Message Function

- *Update the message controller (post function) to include the user id in the request body before saving it(Message.js)*

```
module.exports = {
    …
    post: function (req, res) {

         req.body.user = req.user;

        var message = new Message(req.body);

        message.save();

        res.status(200);
     }
}
```

- *Re-run your server and test it*
  - Post a message while you have a token in your local storage
  - db.messages.find() in mongo console
  - Last messge should have the user id (in **ObjectId**)

Attach User Info to Messages sent back to Front-End

- *Update the message controller (get function) to include the user id in the request body before saving it(Message.js)*

```
module.exports = {
    …
    get: function (req, res) {
        //Attach user to the message and exclude the password
        Message.find({}).populate('user', '-pwd').exec(function (err, result) {
            res.send(result);
        })
     },
}
```

- *Re-run your server and test it*
  - Refresh your message board
  - Check message response inspect → network → message
  - You should find user info attached with the message

Display User Email next to the Message

- *Now, you need to update your front-end ☺*
- *Update your mian.html*
- *Simply add user info beside the message*

```
<ul class="list-group">

   <li class="list-group-item" ng-repeat="message in main.messages">
       {{message.msg}} {{message.user.email}}
   </li>

</ul>
```