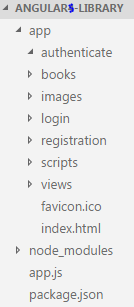
Refactor S07: Organise By Feature

In this section you will create folders by feature and move JavaScript code from **index.html** to a set of newly created JavaScript files.

1. Open previous S07 project in Visual Studio Code (NOTE if you wish to keep previous folder/project simply copy into new folder)
2. Install **nodemon** if you have not done so previously – see last page for details
3. Under the **app** directory create folders **authenticate**, **books**, **login**, **registration**:



1. Move **index.html** into **views** (drag and drop with VSC)
2. Under **app** create file **app.module.js** at (same level as **favicon.ico**)
3. From **index.html** cut all the libraryApp JavaScript (between the <script> …libraryApp... </script>) and paste into **app.module.js**.

Note jshint should kick in now because it is a JavaScript file

1. In **index.html** add the following so the browser knows where to find the libraryApp module

<script src="app.module.js"></script>

1. Check your application is still working by running in browser and you are still getting:
   1. all books
   2. individual books
2. Move **books.html** and **book.html** into **books** folder
3. In **app.module.js** update $routeProvider so templateUrl now points to /books.
4. Again check your app is still working and showing all books and individual books.
5. Within **books** folder create file **booksCtrl.js**
6. At top of **bookCtrl.js** add line

angular.module('libraryApp')

1. From **app.module.js** cut functions controller(‘BookCtrl’,…) & controller(‘BooksCtrl’,… ) and paste into **bookCtrl.js** under line just added in.
2. Modify code so that first few lines are similar to following and showing no error:

angular.module('libraryApp')

.controller('BookCtrl', function ($scope, $sce, $routeParams, libraryQuery) {

…

…

})

.controller('BooksCtrl', function ($scope, libraryQuery) {

…

…

1. Or copy **☺**

angular.module('libraryApp')

.controller('BookCtrl', function ($scope, $sce, $routeParams, libraryQuery) {

// getBook() returns a Promise

libraryQuery.getBook( $routeParams.id )

.then(

(resultsRetuned) => {

$scope.book = resultsRetuned;

$scope.trust = $sce.trustAsHtml;

},

(error) => {

console.log(error);

}

);

})

.controller('BooksCtrl', function ($scope, libraryQuery) {

console.log('BooksCtrl 1 - before async http-get(..)');

libraryQuery.getAllBooks()

.then(

(resultsRetuned) => {

console.log('--BooksCtrl 2 - Fulfilled Promise getAllBooks, $scope updated');

$scope.books = resultsRetuned;

},

(error) => {

console.log('--BooksCtrl 2b - Rejected Promise getAllBooks');

console.log(error);

}

);

console.log('BooksCtrl 3 - leaving');

});

1. Within **books** folder create file **booksFactory.js**
2. Move your factory code from **index.html** or copy following code into **booksFactory.js**

angular.module('libraryApp')

.factory('libraryQuery', function ($http, $q) {

return {

getAllBooks: function() {

console.log('getAllBooks 1 Entered - calling $http-allBooks');

var deferred = $q.defer();

// Get all books from DB - via Express

// $http APIs return a Promise

$http.get('/allBooks')

.then (

(results) => {

console.log('--getAllBooks 2 - results from $http-allBooks - return Promise Fulfilled');

deferred.resolve(results.data);

},

(error) => {

console.log(error);

deferred.reject(error);

});

console.log('getAllBooks 3 Leaving - Promise Deferred');

return deferred.promise;

},

getBook: function (id) {

var deferred = $q.defer();

// Get book from DB - via Express

$http.get('/book/'+id)

.then (

(results) => {

deferred.resolve(results.data[0]);

},

(error) => {

console.log(error);

deferred.reject(error);

});

return deferred.promise;

}

};

});

1. In **index.html** add reference to these scripts (in a suitable place)

<script src="books/booksCtrl.js"></script>

<script src="books/booksFactory.js"></script>

1. Again check your app is still working and showing all books and individual books.

Add a Login Screen

In this section create a login page (inspired from <http://bootsnipp.com/snippets/featured/login-screen-with-background>).

1. In **login** folder create file **login.html** and paste following code

<form ng-submit="submit()" name="register" class="form-signin" novalidate>

<h1 class="form-signin-heading text-muted">Login</h1>

<input name="email" ng-model="email" type="email" class="form-control" placeholder="Email address" required>

<p class="help-block" ng-show="register.email.$dirty && register.email.$invalid">Please enter a proper email.</p>

<input name="password" ng-model="password" type="password" class="form-control" placeholder="Password" required>

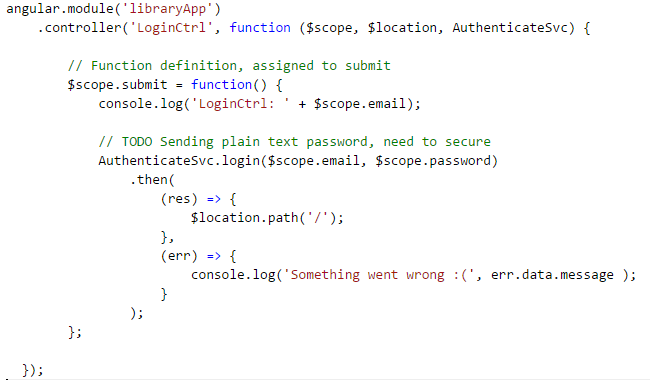
<button ng-disabled="register.$invalid" class="btn btn-lg btn-primary btn-block" type="submit">Login</button>

<br />

<a class="text-muted pull-right" href="#/register">Create account</a>

</form>

1. In **login** folder add new file **loginCtrl.js** and paste following



1. In folder **authenticate** add file **authenticateSvc.js** and paste following code

angular.module('libraryApp')

.service('AuthenticateSvc', function ($http) {

// Login call to express/db

this.login = function (email, password) {

console.log('post login: ' + email);

return $http.post('/admin/login',{email:email, password:password})

.then(

authSuccessful,

(error) => {

console.log('post login error' );

}

);

};

// Register call to express/db for new users

this.register = function (email, password) {

console.log('post register: ' + email);

return $http.post('/admin/register',{email:email, password:password})

.then(

authSuccessful,

(error) => {

console.log('post registration error' );

}

);

};

function authSuccessful(res){

// TODO Set any system variables to deny/allow access to system

console.log('Admin message: ' + res.data.message);

switch ( res.data.message )

{

case 'LOGIN\_SUCCESSFUL':

case 'REGISTRATION\_SUCCESSFUL':

toastr.success( res.data.message, 'Admin message:' );

break;

default:

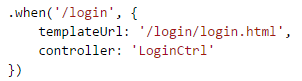
toastr.error('Admin message: ' + res.data.message );

}

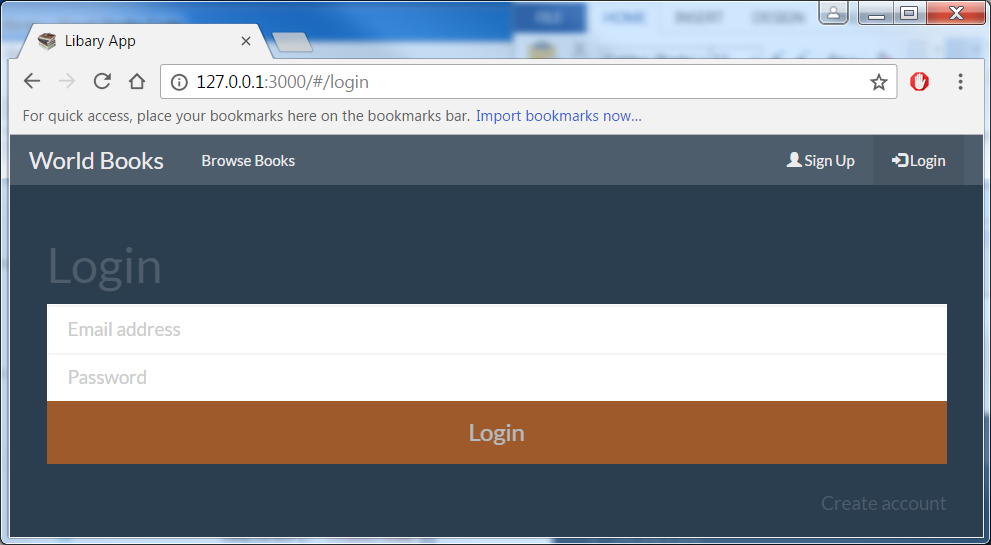
}

});

1. In **index.html** add references to these two new controller JavaScript files.
2. In **app.module.js** add login route



1. Refresh your app and ensure you can see Login page when click Login icon



Add a Login Page & Validation Module

In this section you will create a Login page, an express module which will encrypt/decrypt/verify passwords using external module bcrypt-node and import (require) it into the main app. This module will also save user details to mongoDB.

Also will be using a popular messaging/alert library (toastr, which is configurable too) to show messages on a browser to the user.

1. In VSC create a folder **routes** at same level as **app** folder
2. Inside **routes** folder create a file **admin.js** and paste following login & registration routes:

var express = require('express');

var app = express();

var router = express.Router();

var mongodb = require('mongodb').MongoClient;

var url = 'mongodb://localhost:27017/libraryApp';

var bcrypt = require('bcrypt-node');

// Via route '/admin/...'

router.post('/login', (req, res) => {

req.user = req.body;

//console.log('/login: ' + req.user.email + ', pw: ' + req.user.password);

mongodb.connect(url, (err, db) => {

if ( err ) {console.log('Error connecting to mongoDB', err); return; }

// Connected to DB

var collection = db.collection('users');

// Check login details

collection.find({'email': req.user.email}).toArray((err, docs) => {

// Check if user exist

if ( docs.length === 0 ) {

// User does not exist

db.close();

res.status(200).send({ message: 'LOGIN\_DOES\_NOT\_EXIST'});

} else {

// User does exist in system

// Check if passwords match

// req.user.password is the user entered plain text password

// docs[0].password is the 'hash' (encrypted password) in DB

bcrypt.compare( req.user.password, docs[0].password, function(err,isMatch) {

if (err) { console.log('bcrypt.compare error:'); return (err); }

//console.log(docs);

db.close();

if ( isMatch === true ) {

// console.log('Passwords match');

res.status(200).send({ message: 'LOGIN\_SUCCESSFUL' });

} else {

// console.log('Passwords do not match');

res.status(200).send({ message: 'LOGIN\_PASSWORD\_INCORRECT' });

}

});

}

});

});

});

// Via route '/admin/...'

router.post('/register', (req, res) => {

req.user = req.body;

//console.log('/register: ' + req.user.email + ', pw: ' + req.user.password);

mongodb.connect(url, (err, db) => {

if ( err ) { console.log('Error connecting to mongoDB', err); return; }

// Connected to DB

var collection = db.collection('users');

collection.find({'email': req.user.email }).count( (err,count) => {

// Check if email already registered

if ( count === 0 ) {

// Email not registred, generate hash and save in DB

// Generate Salt for encrypted pw

bcrypt.genSalt(10, (err, salt) => {

if (err) {return(err);}

// From plain text req.user.password create

// hashed password (encrypted password) to save in DB

bcrypt.hash(req.user.password, salt, null, (err, hash) => {

if (err) {return (err);}

// Add new user to DB with hashed password

collection.insert({'email': req.user.email, 'password': hash }, (err,result) => {

console.log('User inserted');

db.close();

res.status(200).send({ message : 'REGISTRATION\_SUCCESSFUL'});

});

});

});

} else {

console.log('Email already exist');

db.close();

res.status(200).send({ message : 'USER\_ALREADY\_EXIST'});

}

});

});

});

// Export so route methods available outside

module.exports = router;

1. In **index.html**
   1. add reference to a popular alert/message box library (after angular-route.js script src)

<script src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>

* 1. and add toastr’s css ref:

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />

1. In **app.module.js** add toastr configuration items in .config (see related url for further info)

// Alert style notification library

// https://github.com/CodeSeven/toastr

toastr.options.timeOut = 4000;

toastr.options.positionClass = 'toast-bottom-right';

1. In the Integrated terminal install the two external modules:

npm install body-parser --save

npm install bcrypt-node --save

1. Back in **app.js** add (before app.use(express.static('app')) ):

var bodyParser = require('body-parser');

var adminRoute = require('./routes/admin');

app.use(bodyParser.json());

// Permissions for Get/Put/Post/Delete from browser

app.use( (req, res, next) => {

res.header('Access-Control-Allow-Origin', '\*');

res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');

res.header('Access-Control-Allow-Headers', 'Content-Type, Authorization');

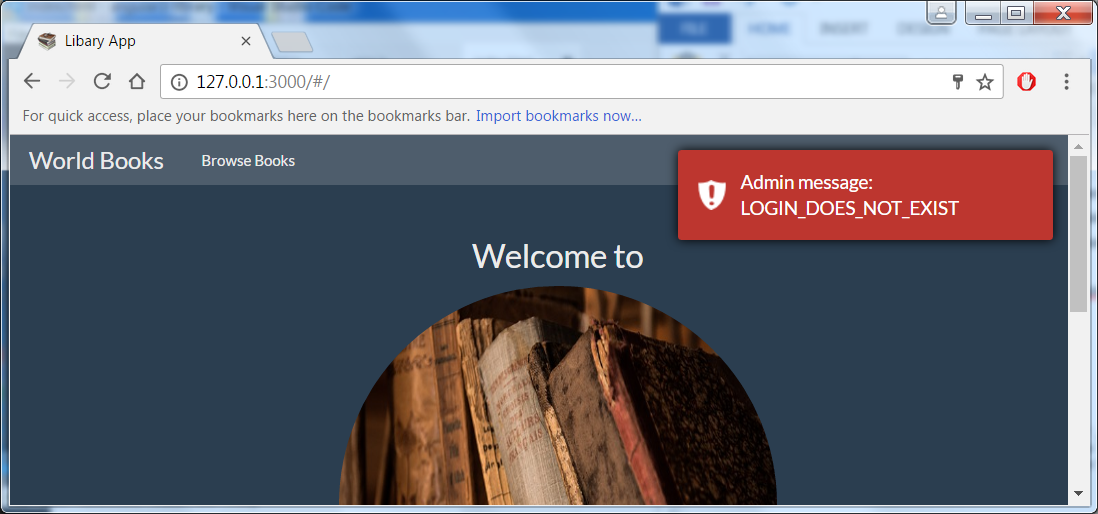
next();

});

// Use adminRoute module to handle /admin calls

app.use('/admin', adminRoute );

1. Try logining in now. Confirm you get LOGIN\_DOES\_NOT\_EXIST message (as collection of users initially empty)



Add a Register Page

On the Login screen there is a link to ‘Create account’ which routes to ‘/Register’, which does not exist. In this section you will create these missing parts and use the validation module.

1. Under **registration** folder create **register.html** and paste following:

<form ng-submit="submit()" name="register" class="form-signin" novalidate>

<h1 class="form-signin-heading text-muted">Register</h1>

<input name="email" ng-model="email" type="email" class="form-control" placeholder="Email address" required>

<p class="help-block" ng-show="register.email.$dirty && register.email.$invalid">Please enter a proper email.</p>

<input name="password" ng-model="password" type="password" class="form-control" placeholder="Password" required>

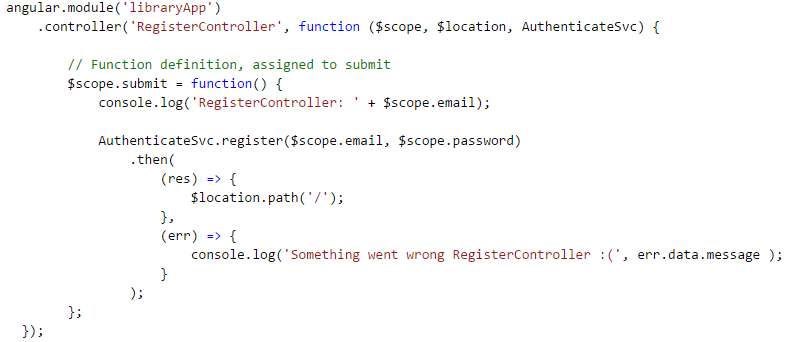
<input name="password\_confirm" ng-model="password\_confirm" type="password" class="form-control" placeholder="Confirm Password" validate-equals='password'>

<p class="help-block" ng-show="register.password\_confirm.$dirty && register.password\_confirm.$invalid">please match the passwords.</p>

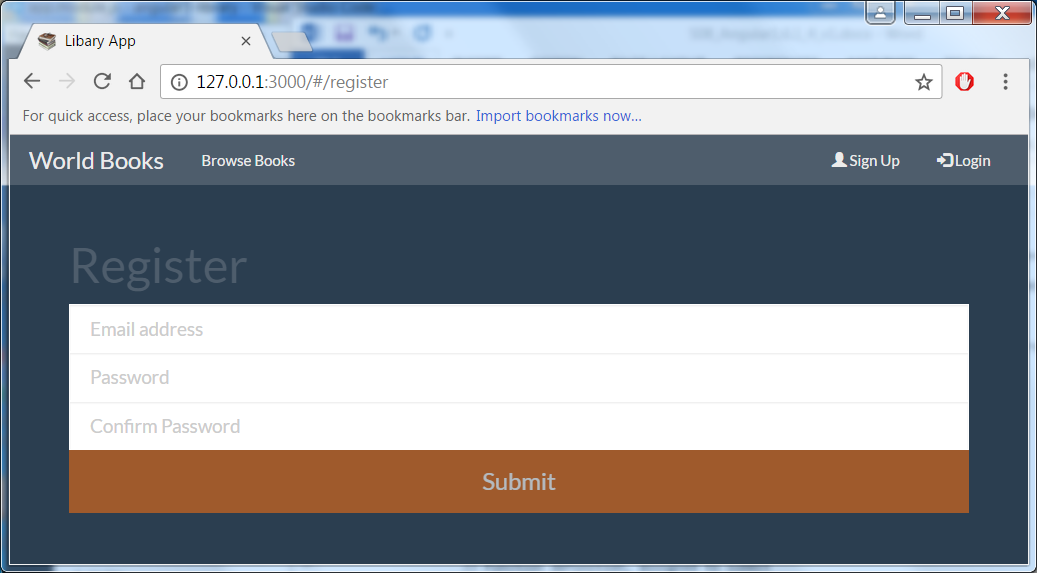
<button ng-disabled="register.$invalid" class="btn btn-lg btn-primary btn-block" type="submit">Submit</button>

</form>

1. Create file **registerCtrl.js** and paste following:



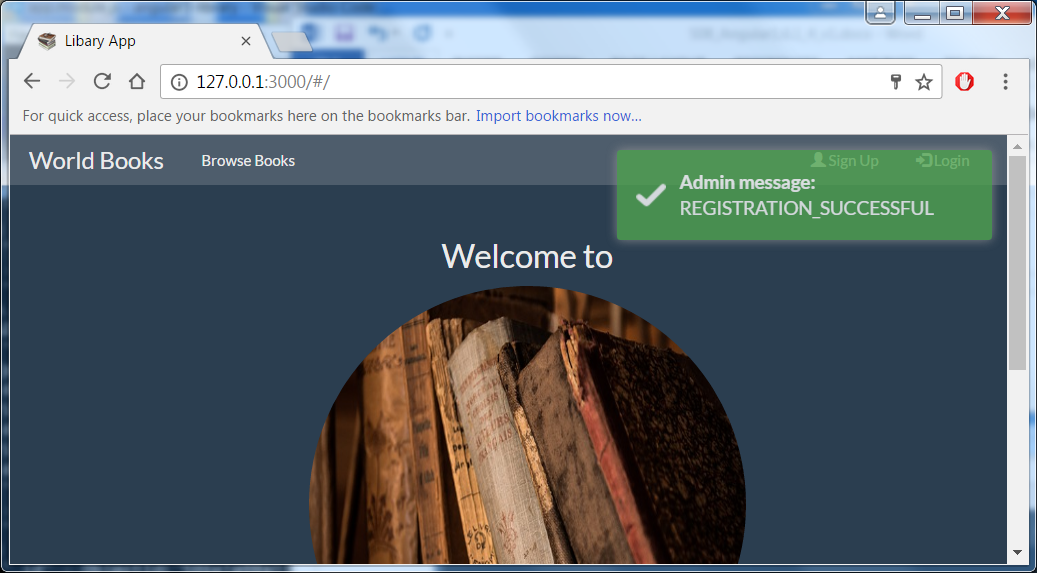
1. In **index.html** add reference to this new Controller JavaScript file
2. Finally to **app.module.js** add route ‘/register’ to $routeProvider.
3. Confirm ‘Create account’ opens Register screen



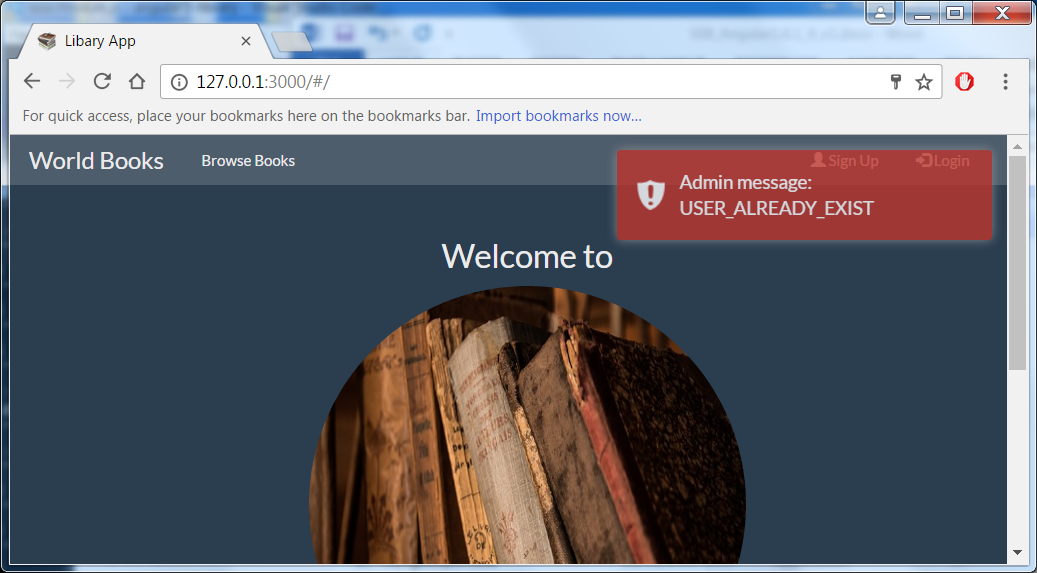
1. Confirm you can Register.

NOTE ‘Password’ & ‘Confirm Password’ do not check they are the same. This will be a TODO.

The ‘Password’ value will be encrypted and saved in ‘users’ collection in libraryApp DB.



1. Try to Register with same email, you should get error message



1. Open a mongoDB shell confirm existence of newly created **users** collection and the encrypted password in the **libraryApp** db.

Hint: list of mongo commands that may be useful:

show collections

show dbs

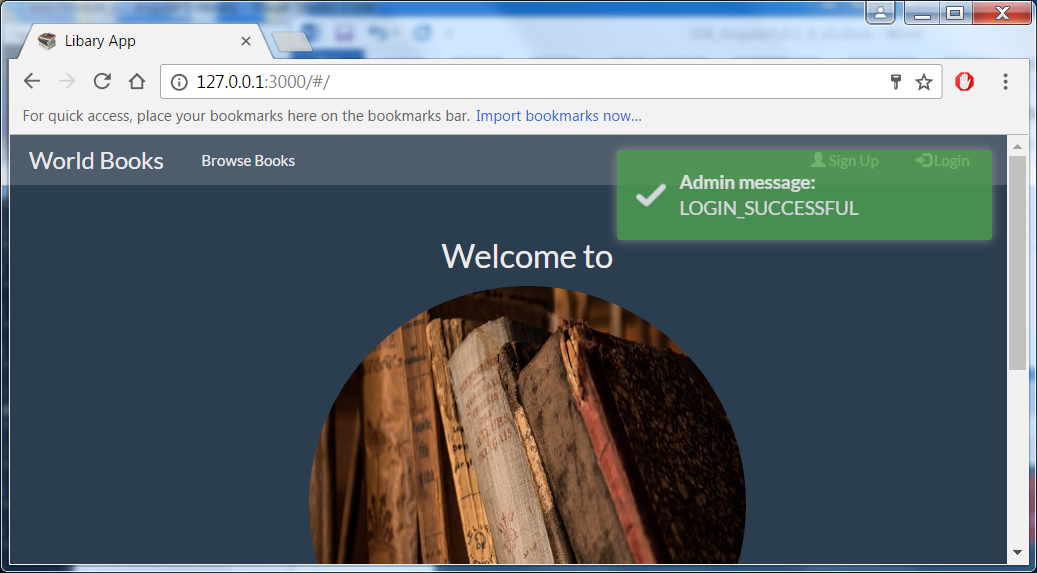
db.users.find()

use libraryApp

use libraryApp

db.users.find().pretty()

1. Now try and login and you should get



TODO

1. Try securing the ‘Browse Books’ link to Logged in users only
2. Change Register screen to ensure ‘Password’ & ‘Confirm Password’ matching

Install nodemon

**nodemon** is a monitoring utility which will automatically restart your application without you having to stop and start your application when you make JavaScript code change.

1. In the Integrated terminal install the **nodemon**:

npm install nodemon –g (-g installs it globally so will be available in other projects)

1. Start your application using **nodemon** rather than **node** so updates will be detected and application automatically restarted. In the Integrated terminal type:

nodemon app (where app refers to your **app.js** application)