**Ctrl+shift+L – bullet shortcut**

Important

-mkdir appname && cd appname

* 1 ‘express’
* 2 ‘npm install’
* 3 ‘npm start’

use nodemon

nodemon –start web at port 3000

rs //to restart the nodemon server

ctrl+c //to stop the server

* 4 create new folder called app\_server
* 5 create 2 new folders models and controllers inside app\_server
* 6 move views and routes to app\_server folder
* 7 in app.js,change to app.set(‘views’, path.join(\_\_dirname,’app\_server’,’views’
* 8 in app.js change var routes= require(‘./app\_server/routes/index’) and same for users
* 9 Split controllers from routes using exports and moving codes.

Adding Heroku

* in package.json add “engines”{ “node”:”version”, “npm”:”version”},
* add procfile- just write ‘web: npm start’
* ‘heroku local’ –start the app locally at port 5000.

To push to heroku

* Git init, git add., git commit –m ‘first commit’
* ‘heroku create’
* Git push heroku master
* ‘heroku ps:scale web=1’ //web dyno
* ‘heroku open’ //open web

--rename app name

* heroku apps:rename newname
* heroku git:clone –a myapp

**mixins**

**Mongodb**

brew services start mongodb

brew services stop mongodb

For versions >4

* var dbURI = 'mongodb://localhost:27017/Loc8r';
* mongoose.connect(dbURI,{ useNewUrlParser: true });

to start mongod server

Closing mongoose connection

* Listen to SIGINT and SIGUSR2 or SIGTERM for Heroku.
* Use process.once(signal, function());

For opening connections to multiple dbs

* Use createConnection

Var dbURIlog = ‘mongodb://localhost/Loc8rLog’;

Var logDB = mongoose.createConnection(dbURIlog);

* And logDB.on(‘connected’, function(){…}; to monitor connection

Node

* ‘npm start’ runs ‘node server.js’ command.
* ‘npm –g install supervisor’ – this module auto restarts the server when changes are saved. Start server by ‘supervisor serjer.js’

Representational State Transfer:REST API

* REST is stateless- no idea of current user state or history.
* Abbreviation for application program interface, which enables applications to talk to each other.
* In MEAN, it is used to create a stateless interface to database, enabling a way for other applications to work with the data.

Problems with Single Page Application(SPA)

* Crawlability: js are difficult to crawl. Sols-1 have server create HTML-based version and deliver that to crawler. 2 use headless browser such as PhantomJS to run JS.
* Analytics integration: analyse the traffic, can be added using different APIs
* Speed of initial load: need to load the framework at least once and all the contents at the beginning. To overcome: use lazy loading.

Blog example

* Has two sides Blog entries- needs to be fast(pages changes instantly) and admin interface- needs to be interactive. Soln is to create admin interface with angularjs and blog entries with express & nodejs with angularjs for adding features.

**Prototype development stages:**

1. build static site
2. design data model and create database
3. build data API
4. hook database into the application
5. augment the application.

Express

* Express is a web application framework for no.

Package.json file

* In every node app there should be package.json in root folder.
* It contains metadata about the project including packages that it depends on.
* ‘npm install’ run It in same folder as package.json to install all the dependencies listed in the file.
* All packages are installed in node modules, inside subfolders and they all have package.json file each.
* To add more packages: ‘npm install –save package-name’ run it in the same folder as package.json.

**Express project**

* You will need: node &npm, express generator installed globally, git , heroku and terminal.
* Verify installation: express –-version.
* 4 template engines available: Jade is default, EJS, JSHtml and Hogan.
* About Jade: no <> except for div, no tag means <div>. indented using spaces. E.g., #banner.page-header - # for id and . for class so it means id=”banner” and class=”page-header”

**Creating Express project:**

* Go to the project folder and type ‘express’-will create folders then go to folder where package.json is and enter ‘npm install’-installs dependencies
* Try it out: ‘npm start’ and in browser goto ‘localhost:3000’.
* **Nodemon** to auto restart the server when code changes. ‘npm install –g nodemon’ and go to the project folder and ‘nodemon’.
* In the root folder:
* Create new folder ‘app\_server’ and inside it-
* Move ‘views’ and ‘routes’ folders
* Create 2 more folders ‘models’ and ‘controllers’
* To let express know where ‘views’ have moved- open app.js and find the line

app.set(‘views’, path.join(\_\_dirname, ‘views’)); and change it to

app.set(‘views’, path.join(\_\_dirname, ‘app\_server’, ‘views’));

* To let express know where ‘routes’ is, in app.js find the lines

var routes = require(‘./routes/index’);

var users = require(‘./routes/users’);

And change them to –

var routes = require(‘./app\_server/routes/index’);

var users = require(‘./app\_server/routes/users’);

* **Separate controllers from routes** : in default express controllers are part of routes.
* Inside index.js in ‘routes’-

router.get(‘/’, function(req, res){

res.render(‘index’, {title:’Express’});

});

1) **in the same file but outside route**:

var homepageController = function(req, res){

res.render(‘index’, {title:’Express’});

then,

router.get(‘/’, homepageController);

2) **in separate file**:

‘exports’ make functions available to other files.

module.exports.index = function(req, res){

res.render(‘index’, {title:’Express’});

};

then in routes file

var ctrlMain = require(‘../controllers/main’);

router.get(‘/’. ctrlMain.index); //index is the function required.

* Using Bootstrap in the project
* download bootstrap and unzip
* create a new folder in express project called ‘bootstrap’ and copy the unzipped file.
* it also needs jQuery, download it and save it in public/javascripts.
* reference bootstrap.min.js, jquery in the ‘layout.jade’ file.
* delete /public/style.css file
* Heroku for deploying the server.
* Heroku needs to know the version of node and npm, so add following to package.json

“engines”:{

“node”: ”~4.2.1”, //current version “node –version”

“npm”: “~2.2.0”

},

* Create a Procfile:
* it tells how to start node app.
* in root folder create “Procfile”, no extension and enter “web: npm start
* Heroku comes with Foreman to test it locally before pushing it to Heroku. Stop the app first and enter ‘foreman start’. port is 5000.
* use git to push the app to Heroku.
* git init
* git add .
* git commit –m ‘first commit’
* Create Heroku App:
* ‘heroku create’
* git push heroku master
* web Dyno
* more dynos -> more resources for your app. good for big app
* ‘heroku ps:scale web=1’ and now the url is live.
* to view the app : ‘heroku open’
* to update use git.

**4 Building Static site with node and express**

**Creating routes**

* define all the routes in the index.js file as follows:

var ctrlLocation = require(‘../controllers/locations’);

var ctrlOthers = require(‘../controllers/others’);

router.get(‘/’, ctrlLocations.homelist);

router.get(‘./location’, ctrlLocations.locationInfo);

…

…

module.exports = router;

**Create Controllers**

* remove any default files in controllers folder that we don’t need.
* create others.js controller in the folder and add:

module.exports.about = fuction(req,res){

res.render(‘index’, {title: ‘About’});

};

* here ‘index’ is the view.

**Create Views**

**MongoDB**

Install mongodb: brew update && install mongoDB

mongod --version //check if installed properly

npm init //to start a new project

npm install mongodb - -save // to add dependencies

e.g.,

var MongoClient = require(‘mongodb’).MongoClient;

var connectionUrl = ‘mongodb://localhost:27017/myproject’,

sampleColleciton = ‘chapters’; //myproject db will be created.

//data to insert to db

var chapters = [{

‘title’ : snow crash’,

‘author’: ‘neal stephenson’

},{

‘title’ : snow crash’,

‘author’: ‘neal stephenson’

}];

MongoClient.connect(connectionUrl , function(err,db){

console.log(‘connected correctly’);

var collection db.collection(sampleCollection);

collection.insert(chapters, function(error, result){

if(!error){

console.log(‘success’);

}else{

console.log(‘error’);

}

db.close();

});

});

* save it as app.js
* ‘mongod’ to start the Mongodb demon
* node app.js to run the file.

Checking the db:

* mongo
* use myproject
* show collections // shows collections in db
* db.chapters.find().pretty() // displays db

Mongodb basics:

* mongo
* show dbs // show list of dbs
* use chapter3 // switched to db chapter3
* show collections // shows collections in that db

Inserting data to db

* db.newCollection.insert({name:’hem’, website:’lallu lallanwa’})
* db.newCollection.find().pretty() // shows the collection

Query

* db.newCollection.find({name:’hem’})
* db.newCollection.find({name:’hem’}, {name:true}) // returns only \_id and name
* db.newCollection.find({name:’hem’}, {name:true, \_id:false}) // returns only name
* db.homeword\_scores.find({score:{$gte:80,$lt:90})
* db.newCollection.find({name:{$regex:’hem’}})

Updating data

* db.newCollection.update({name:’hem’},{$set:{website:’pun’}})
* db.newCollection.update({name:’joe’},{name:’joe’,website:’dfds’},{upsert:true}) //if data does not exist, it add to db

Deleting data

* db.newCollection.remove({name:’joe;})
* db.newCollection.drop() // delete all data in a collection

**Naming Convention**

* each entry in database is called a document
* collection of documents is a collection-like table
* definition of a document is schema
* data entity defined in a schema is a path.

Schema

it is a JS object.

{

firstname : {type:String, required:true},

surname : {type:String},

rating: {type: Number, “default”:0, min:0, max:5}

}

inside schema file:

require(Mangoose);

var locationSchema = new mangoose.Schema({ //schema goes here });

* nested schema is supported- openingTimes:[openingTimeSchema]

Compiling Mongoose schemas into models

mongoose.model(‘Location’, locationSchema, ‘locations’);

connectionName.model(nameOfModel,SchemaToUse,CollectionName(optional));

add at the end of schemas

* mongoose.model(‘Location’,locationSchema);

**Creating MongoDB database:**

>mongo

>use Loc8r

>show dbs

>db.locations.save({

…..

)}

>show collections

>db.locations.find().pretty()

>db.locations.update({……..})

**Pushing db online**

* signup for free acc on mlab
* create new database
* add user
* get database URI
* set the MONGODB\_URI via settings in heroku website check-heroku config:get MONGOLAB\_URI
* create tmp folder to dump db and dump it using ‘mongodump –h localhost:27017 –d Loc8r –o ~/tmp/mongodump’
* restore db to live db using ‘mongo restore –h <hostname> -d <dbname> -u <uname> -p <pwd> <dumplocation>

Edit code to select local and online db properly using NODE\_ENV,

**Adding Mongoose to app**

* npm install –save mongoose //creates new mongoose folder in node\_modules and is added to dependencies

create db.js in app\_server/models. add var mongoose=require(‘mongoose’);

import it in app.js. ‘require(‘,/app\_server/models/db’);’

creating mongoose connection

database URI has following construct

mongodb://username:pasword@localhost:27027/database

server address/port/databasename

* add following to db.js

var dbURI = ‘mongodb://localhost/dbname’; // uname and pwd are optional

mongoose.connect(dbURI);

mongoose.connection.on(‘connected’,function(){

console.log(‘Mongoose connected to ‘ + dbURI);

});

* copy code from page 129 MEAN book for db.js

**installing modules using npm**

npm install express

var express = require(‘express’);

* installing module globally

npm install –g express // creates node\_modules directory in system directory

* npm install express - -save //dependencies- necessary for app
* npm install express - - save-dev //dev dependencies only necessary for development of app.

**REST API ()**

Representational State Transfer.

* collections in database has a set of API URLs.

**Request methods**

* POST : create a new data in database
* GET: read data from database
* PUT : update a doc in db
* DELETE : delete an object from db

API should return one of three:

* a JSON object containing data answering the request query
* a JSON object containing error data
* a null response

**Creating Routes**

* create file index.js in app\_api/routes
* To app.js file add

var routesApi=require(path to js file)

app.use(‘/api’, routesApi);

**setting up request methods in routes**

router.get(‘/location’, ctrlLocations.locationInfo);

router.post(‘/location’, ctrlLocations.locationsCreate);

**routes in page 169 of MEAN**

//locations e.g.,

router.get(‘/locations’,ctrlLocations.locationsListByDistance);

….

//reviews

router.post(‘/locations/:locationid/reviews’, ctrlReviews.reviewsCreate’);

router.get(‘/locations/:locationid/reivews/:reviewid’,ctrlReviews.reviewsReadOne);

…..

module.exports = router;

**Creating the controller**

* 2 files ‘locations.js’ and ‘reviews.js’ in app\_api/controllers folder

**Including the model**

var mongoose = require(‘mongoose’);

var Loc = mongoose.model(‘Location’);

**get method: read data from MongoDB**

* code in page 177 of MEAN

**Find a single document in MondoDB**

* mongoose query methods
* find, findById, findOne, geoNear, geoSearch
* e.g., Loc.findById(locationid)
* exec method to start the query.
* E.g., Loc.findById(locationid).

.exec(function(err, location){

console.log(‘findById complete’);

});

* select() to select specific subdocument
* e.g., Loc

.findById(req.params.locationid)

.select(‘name reviews’)

.exec(function(err, location){

console.log(‘findById complete’);

});

* findById() , geoNear(), etc
* e.g., Loc.findById(req.params.locationid)

.exec(function(err, location){ //**location** is instance of found obj

location.name = …

* code in page 177 of MEAN
* geoNear() is a model method that finds list of locations close to a specific point
* it has three para(point, options, callback);
* para point is a geoJSON point. e.g., var point = {type:”point”, coordinates: [lng,lat]};
* options e.g., var geoOptions={spherical:true, maxDistance: theEarth.getRadsFromdistance(20), num: 10}; // geoNear only takes radians so convert distance to rads and ‘num’ lists only 10 closest locations
* the output has three parameters: error obj, results obj and stat obj.
* e.g., Loc.geoNear(point, options, function(err, results, stats){}; //results holds the returned result value.

**POST methods: Adding data to MongoDB**

* Loc.create(dataToSave, callback);
* for sub docs: once subdocuments have been added using create() the parent document must be saved using save() method.
* e.g., location.save(function(err, location){}); //**location is returned data**
* form data in req.body. e.g., location.name = req.body.name

**Put methods: Updating data from MongoDB**

* Post creates new data each time, Put does not.
* use findById to find the data and make changes and then save it.
* form data in req.body. e.g., location.name = req.body.name
* .select(‘-reviews –rating’) selects all but these two.

**Delete method: deleting data from MongoDB**

* findByIdAndRemove()
* or
* find the data and delete it: Loc.findById() and Loc.remove()
* if deleting subdocument , save the parent doc after deleting.

**Consuming a REST API**

Call API from Express

* we need to use **request** package
* npm install –save request
* at the top of /app\_server/controllers/locations.js add

var request = require(‘request’);

* add apiOptions=URL on locations.js. –page 204 either local or online depending on the NODE\_ENV
* request(option, callback) //callback :function to call when response is received.
* request options – url, method(get,post,put or delete), json(body), qs(query string)
* e.g.

var requestOptions={

url: <http://fjsdlfjsd>,

method: “GET”,

json:{},

qs:{

offset:20

}

};

* request callback has three para(err, full response, parsed body of response)
* e.g.,

function(err, response, body){

if(err){ console.log(err);}

else if(response.statusCode === 200) {console.log(body);}

else{console.log(response.statusCode)}

};

**Using data from API**

* current homepage controller only renders the hard-coded date to view. But we want to render after data is returned by the API from database.
* **Move rendering to named function**
* make a function called **renderHomepage** in appserver/controllers/locations.js
* Getting data by asking the API

Data Validation

**At Schema level with Mongoose**

* make name and review text required at schema.

**At application level with Node and Express**

* check submitted data before sending it to API

**In browser with jQuery**

**AngularJS**

Use $scope only when absolutely necessary, else use controllerAs : ‘vm’.

**To speed up the angularjs app**

* **IIFE: Immediately Invoked Function Expression**

(function(){

console.log(“Output immediately”);

})();

* **Minify the code**
* manually injecting dependencies to protect against minification
* use UglifyJS to minify and concatenate scripts

Tell nodemon to ignore changes in the public to prevent from restarting again and again

{

“verbose”: true,

“ignore”:[“public/\*”]

}

* remove # from the address location by turning on the html5 mode

$locationProvider.html5Mode(true);

ngSanitize to let subset of html to be displayed correctly in the browser.

**Using AngularUI components (Modal in our case)**

* create modal that will pop up and user can enter data there.
* download angular-ui –all or single component by clicking ‘create a build’
* unzip downloaded file and save to app\_client/lib and reference them
* add as dependency in app.js just like ui-route, ngSanitize, etc.
* inject them in the controller where it is used
* Define modal instance

vm.popupReviewForm = function(){ //function invoked by button

var modalInstance = $modal.open({

templateUrl: ‘/reviewModal/reviewModal.view.html’ //view to popup

controller: ‘reivewModalctrl as vm’

});

};

* Passing data to modal instance
* resolve variables in the modal instance

get varibles into modal instance def by using ‘resolve’ option.

para should be mapped to a function that returns single value or obj.

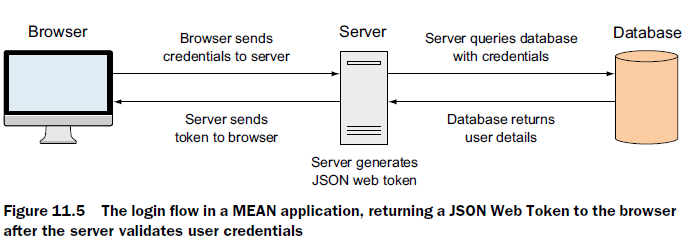
* inject these resolved variables as dependencies into modal controller

as before and also save them as property of the modal

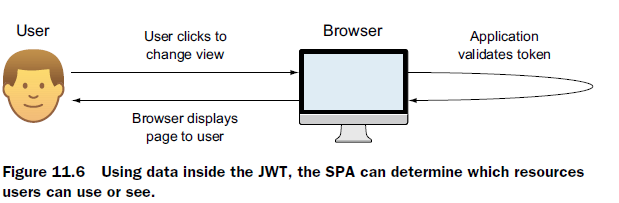
* map them to objects in modal view

**Managing Authentication and user sessions**

The validation process:

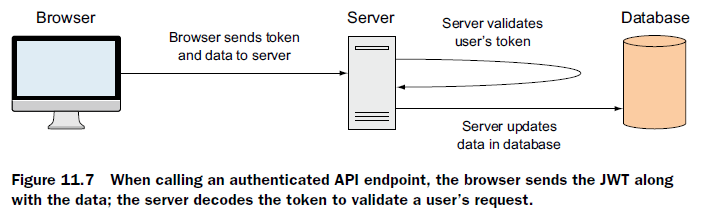


* the browser sends the user credentials to server – ver queries database with credentials – database returns user details – Server generates JWT tokens – server sends token to browser



* session is stored in browser. both server and browser stores JWT token for future use.

Changing views during authenticated session



* user clicks change view – application validates token – browser displays page to the user.

Calling API for updates

* browser sends token and data to server – server validates token – server updates data in database.

Save passwords after encryption.

* need model for user. make userSchema

var userSchema = new mongoose.Schema({

email:{type:String, unique:true, required:true},

name:{type:String, required:true},

hash:String,

salt:String

});

* add methods to schema so that properties can be accessed

**Crypto module**

* two methods: randomBytes – to generate salts and pbkdf2Sync – to generate hash from pwd and salt.
* require crypto.

this.salt = crypto.randomBytes(16).toString(‘hex’);

this.hash = crypto.pbkdf2Sync(password, this.salt, 1000, 64).toString(‘hex’);

* to validate generate new hash and compare it with this.hash

JSON Web Token(JWT) pronounced jot

* used to pass data around in between spi in server and spa in browser
* JWT is a random-looking dot-separated strings
* has three parts: Header, Payload and Signature
* install jwt generating module

**npm install jsonwebtoken –save**

* require’…’
* ‘sign’ method to generate JWT in schema method.
* create environment variables to set secret code
* in root folder create .env and set JWT\_SECRET=this.IsSecret
* add it to .gitignore
* to use env install dotenv module as ‘npm install dotenv –save’ and require as require(‘dotenv’).load();
* and process.env.JWT\_SECRET to use
* set variable for production in hereku

**Creating an Authentication API with Passport**

npm install passport –save

npm install passport-local –save

* create /app\_api/config/passport.js
* to set passport strategy

passport.use(new LocalStrategy({

usernameField: ‘email’}, //username is default, replace username by email

function(~~username~~ username, password, done){}

));

**Secure the relevant API endpoints**

* only allow users with valit JWT to call new review API
* inside controller validate that user exists and can create review
* use middleware express-jwt package

**Managing a user session in Angular**

* use Local storage to store JWT
* Create service to save and reas JWT in local storage