Course name: Web Development with ExpressJS

The sections on uploading and serving files is awesome

===

It seems you can create a starter project with "express -s" in command line (the -s is for session support).

===

In WebStorm, you can create a project of project type "Node.js Express App". When you do this, you actually get a run configuration set up (press the start button in the toolbar).

Check Express.js samples on github.com/visionmedia/express

Search on npm.org before you re-invent the wheel

===

WebStorm has a feature called live edit, can use to see CSS changes to a page without refreshing!

WebStorm also appears to auto-save files at every change!

===

Basic architecture: Request, routing, function, view.

A request comes in, and is processed by a routing mechanism that parses out what function has to be called. Once this function is called, it then provides a response. Most of the time this response is a webpage (the view) but it can be anything (downloadable file, xml, json, etc).

Express.js has no concept of a controller. Can think of Express as having a Model-View-Router.

===

Do not need to do "npm init" in order to do npm installs!

To install an older version of a module:

npm install express@3.X

===

node.js allows javascript to run on the server side

In cmd, if you simply type in "node" + enter, you can now type javascript into the command line!

Can also execute javascript files by typing "node filename.js" into the command line

===

\*\*\*\*\*\*\*\*\*\*

myUtils.js

\*\*\*\*\*\*\*\*\*\*

var printMessage = function(message) {

console.log("Message: " + message);

};

exports.printMessage = printMessage;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

usingMyUtils.js

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var myUtils = require("./myUtils.js");

myUtils.printMessage("Bla bla bla");

---

Think of "myUtils" as a module and "usingMyUtils" imports this module. What you write onto exports are the variables & functions that you'll have access to using myUtils.<function name>

===

node includes by default a module called fs (short for filesystem)

var fs = require("fs");

nodejs uses a callback pattern. This is why most node functions have a callback function as the last argument.

nodejs functions have two versions: the default (asynchronous) version and the synchronous version.

You can see this in the documentation for the fs module:

nodejs.org/api/fs.html

Ex/

fs.unlink(path, [callback])

fs.unlinkSync(path)

The callback function is always executed after whatever asynchronous method it's attached to has been performed.

===

The nodejs equivalent of "hello world": Create a simple http server!

\*\*\*\*\*\*\*\*\*

server.js

\*\*\*\*\*\*\*\*\*

var http = require("http");

var server = http.createServer(function(request, response) {

response.writeHead(200, { "Content-Type": "text/plain" });

response.write("This is a simple HTTP Server");

response.end();

}).listen(3000);

In cmd, do "node server.js" and in a browser type localhost:3000.

(Actually better to use "nodemon server.js" to avoid having to restart the server for changes)

===

Express.js is a Sinatra-inspired framework

It is built on top of Connect middleware. The middleware provides a layer of functionality on top of the native node http server.

===

The hello world for express is even simpler than for node!

\*\*\*\*\*\*\*\*

index.js

\*\*\*\*\*\*\*\*

var express = require("express");

var app = express();

app.get("/", function(req, res) {

res.send("Welcome to express");

});

app.listen(3000);

>> In cmd: nodemon index.js

>> In browser: localhost:3000

===

The "naive" way of making routes:

\*\*\*\*\*\*\*\*

index.js

\*\*\*\*\*\*\*\*

var express = require("express");

var app = express();

app.get("/", function(req, res) {

res.send("Welcome to express");

});

app.get("/customers", function(req, res) {

res.send("Welcome to customer section");

});

app.get("/customers/create", function(req, res) {

res.send("Welcome to customer creation section");

});

app.listen(3000);

===

home = require("./routes/home.js")

The dot slash (./) just means the current directory.

===

A way to refactor the routes for more modularity:

\*\*\*\*\*\*\*\*

index.js

\*\*\*\*\*\*\*\*

var express = require("express"),

home = require("./routes/home.js"),

customer = require("./routes/customer.js");

var app = express();

app.get("/", home.index);

app.get("/customer", customer.index);

app.get("/customer/contact", customer.contact);

app.listen(3000);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

routes/home.js

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

exports.index = function(req, res) {

res.send("Welcome to the index home page");

};

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

routes/customer.js

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

exports.index = function(req, res) {

res.send("Welcome to the customer page");

};

exports.contact = function(req, res) {

res.send("Welcome to the customer contact page");

};

===

To handle variable url paths:

app.get("/customer/:id", function(req, res) {

res.send("Customer selected is " + req.params.id);

});

Works the same way as in Angular routing

===

To handle queries in the url:

app.get("/customer", function(req, res) {

res.send("Customer selected is " + req.query.id);

});

In the browser: localhost:3000/customer?id=300

===

You can actually put in regular expressions into app.get().

===

\*\*\*\*\*\*\*\*

index.js

\*\*\*\*\*\*\*\*

var express = require("express");

var app = express();

app.configure(function() {

app.set("view engine", "jade");

app.set("views", \_\_dirname + "/views");

app.use(express.static(\_\_dirname + "/public"));

});

app.get("/", function(req, res) {

res.render("index");

});

app.listen(3000);

---

When you navigate to the root of the website, res.render() will render the index.jade file found in the views folder.

===

Awesome way to render all your views:

app.get("/:viewname", function(req, res) {

res.render(req.params.viewname);

});

===

jade's page templating is pretty cool

Simply make a layout.jade page that ends with "block content".

Then on any other jade page, write "extends layout" followed by "block content" followed by the actual page content.

jade file watcher (in WebStorm)

===

jade's stylus an alternative to css

jade has a simple looping syntax

each <item> in <customers>

<nested stuff here>

===

When doing a POST request from a form, req.body gives you all the form information that was sent (ex/ req.body.name, req.body.email, etc).

This only works if you have "app.use(express.bodyParser());" inside of your app configuration.

===

Need to add the middleware "app.use(express.methodOverride());" in order for the verbs put and delete to function as normal (at least in a form?)

===

Setting, checking and clearing cookies is trivially easy

\*\*\*\*\*\*\*\*\*

cookies.js

\*\*\*\*\*\*\*\*\*

var express = require("express");

var app = express();

app.use(express.cookieParser("This is my passphrase"));

app.get("/", function(req, res) {

   if(req.cookies.beenHereBefore == "yes") {

       res.send("You have been here before");

   } else {

       res.cookie("beenHereBefore", "yes");

       res.send("This is your first visit");

   }

});

app.get("/clear", function(req, res) {

   res.clearCookie("beenHereBefore");

   res.redirect("/");

});

app.listen(3000);

cmd> nodemon cookies.js

browser> localhost:3000

You can then check your cookies under Resources -> Cookies -> localhost (in Chrome dev tools)

===

Can also use express to handle sessions

\*\*\*\*\*\*\*\*\*\*

session.js

\*\*\*\*\*\*\*\*\*\*

var express = require("express");

var app = express();

app.use(express.session({secret: "This is my secret" }));

app.get("/", function(req, res) {

   req.session.name = req.session.name || new Date().toUTCString();

   console.log(req.sessionID);

   res.send(req.session.name);

});

app.listen(3000);

cmd> nodemon session.js

browser> localhost:3000

Can use redis.io for persisting session info to a database

===

Single vs global interceptors

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interceptor.js

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var express = require("express"),

   util = require("util");

var app = express();

var globalInterceptor = function(req, res, next) {

   console.log(util.format("Path requested is %s", req.path));

   next();

};

var singleInterceptor = function(req, res, next) {

   console.log("single interceptor called");

   next();

};

app.use(globalInterceptor);

app.get("/", function(req, res) {

   res.send("Default is called");

});

app.get("/other", singleInterceptor, function(req, res) {

  res.send("Other is called");

});

app.listen(3000);

===

Check the section on "Authentication and Authorization" for a way to make private webpages. He basically creates middleware that redirects you from a private page to a login page if you're not authenticated. I'm curious if someone could bypass this by disabling javascript on the page.

===

app.use(app.router) checks if a verb request will execute. If not, then call the interceptor functions called by app.use() after app.router.