# Node.js, Express, Jade, and MongoDB

## Getting Set Up

1. Express is an MVC framework. Install express:  
   $[sudo] npm install –g express
2. Install the express generator, which will create a skeleton for your application.  
   $[sudo] npm install –g express-generator
3. Navigate to the directory that will contain your project directory. We will use the express generator to create the directory and the site scaffolding.  
   $ express demo  
   View the directory contents. You should see app.js (the file that runs the server), package.json, a routes directory, and a views directory, and some other stuff.
4. Open package.json and add dependencies: mongodb and monk. Asterisk means “get the latest version.” Don’t forget to add a comma at the end of “serve-favicon.” Monk is a library that works with MongoDB; it makes MongoDB easier to work with. (See <http://monk.readthedocs.org/en/latest/faq.html> for more.)  
     
   $ vi package.json (Esc + I to edit, Esc + :wq! to save and quit)  
     
   {

"name": "mongo",

"version": "0.0.0",

"private": true,

"scripts": {

"start": "node ./bin/www"

},

"dependencies": {

"body-parser": "~1.10.2",

"cookie-parser": "~1.3.3",

"debug": "~2.1.1",

"express": "~4.11.1",

"jade": "~1.9.1",

"morgan": "~1.5.1",

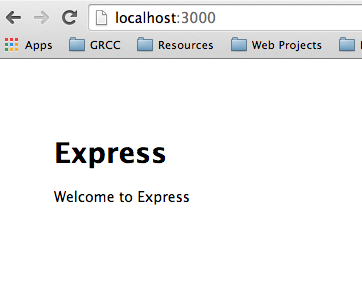
"serve-favicon": "~2.2.0"**,**

**"mongodb": "\*",**

**"monk": "\*"**

}

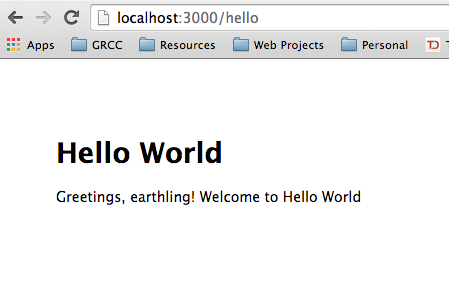
}

1. Install the modules at the project level.   
   $ [sudo] npm install  
   View the directory contents, and the contents of node\_modules.
2. Test your web server.   
   $ npm start  
   This runs your app.js file, which starts your server.
3. Visit localhost:3000 in a Web browser. You should see this:  
      
   You are running Node JS server with the Express engine and Jade HTML preprocessor!
4. Directives tell Express which route files to use. Open demo/routes/index.js. It requires Express, then defines a router variable that points to the express object’s Router method. It uses that method when a page is accessed (router.get). The router function is exported back to the app through module.exports.  
     
   There is already a route for the index page (‘/’). It renders a response using the “index” view with a title of “Express.” Open demo/views/index.jade. The title that was set in index.js is referenced in index.jade. Add a paragraph to index.jade, then refresh the browser. (You don’t need to stop and restart the server when you change a view.)  
     
   p It’s nice to meet you.
5. We’re going to add a new “hello” route. Go to index.js. Copy the router.get and modify it:  
     
   /\* GET hello world page. \*/  
    router.get('/hello', function(req, res, next) {  
    res.render('helloworld', { title: 'Hello World' });  
   });
6. We will use Jade to create a page for res.render to render to! Jade is a templating language used to access our Node/Express data. It uses indentation to indicate structure, rather than close tags. Our Jade files are saved in the views directory. Open demo/views/index.jade and save it as helloworld.jade. Modify the file:  
     
   extends layout

block content

h1= title

p Greetings, earthling! Welcome to #{title}

1. Stop (Ctrl+C) and start (npm start) the server. Then, navigate to <http://localhost:3000/hello>. hello is the route that gets rendered by helloworld.jade.  
     
   
2. See if you can add a list to your hello page using <http://jade-lang.com/reference/tags/>
3. Add a hyperlink using <http://jade-lang.com/reference/attributes/>
4. Add another route to your index.js file, and another jade template to render that route. Remember, you don’t have to stop and restart the server when you change a jade file.

## Displaying data from the database in a web page

1. Stop your server, and create a demo/data directory.  
     
   $ mkdir data
2. Open a new terminal window. Navigate to your mongodb directory and start your database server:

On a Mac:  
$ cd /Applications/MongoDB  
$ mongod --dbpath /Users/laptop/node/328/demo/data

On a PC:  
$ cd C:\MongoDB  
$ mongod --dbpath c:\node\demo\data  
  
It should say:   
[initandlisten] waiting for connections on port 27017

1. Open a third terminal window and type mongo. Create a database, and add some users:

use dbtest

db.users.insert({"username":"tostrander", "email":"tostrander@greenriver.edu"})

newusers = [{ "username" : "jshmo", "email" : "jshmo@msn.com" }, { "username" : "jgarcia", "email" : "jgarcia@comcast.net" }]

db.users.insert(newusers);  
db.users.find().pretty()

1. Now, we want to display our data from the database in a web page. Open app.js. This file is importing various modules and assigning the objects they return to variables, and creating variables for routes and users. Routes are a combination of models and controllers – they direct traffic. Express created these paths.

//Instantiates Express and assigns the object to app.  
var app = express();   
  
The next section uses the app variable to configure Express, including where to find views, and which engine to use to render the views (Jade).  
  
app.use(express.static(path.join(\_\_dirname, 'public')));  
  
This tells Express where to get static files (/public/) but makes it seem like they’re coming from the top level. So c:\node\mongo\public\images will be accessed view http://localhost:3000/images.  
  
Then we have error handlers for 404 errors, and for development and production. (In dev mode, you will get more information.)  
  
//Export the app object, so that it can be used elsewhere

module.exports = app;  
  
Add three lines to app.js:

var express = require('express');

var path = require('path');

var favicon = require('serve-favicon');

var logger = require('morgan');

var cookieParser = require('cookie-parser');

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

**var mongo = require('mongodb');**

**var monk = require('monk');**

**var db = monk('localhost:27017/dbtest');   
//dbtest is the name of our db. 27017 is the default port //that MongoDB runs on.**

1. Next, we need to make our database accessible to our router. Still in app.js, add this code *before* the other app.use statements, so that those routes can use the database.

**// Make our db accessible to our router**

**app.use(function(req,res,next){**

**req.db = db;**

**next();**

**});**

app.use('/', routes);

app.use('/users', users);

1. Next, we will create a route that displays data from the database. Open demo/routes/index.js and add a new route:

/\* GET Userlist page. \*/

router.get('/userlist', function(req, res) {

var db = req.db;

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

collection.find({},{},function(e,docs){

res.render('userlist', {

"userlist" : docs

});

});

});

This is taking the db object we passed to the request, and using it to fill the “docs” variable with user data.

1. Open demo/views/index.jade and save it as userlist.jade.

extends layout

block content

h1.

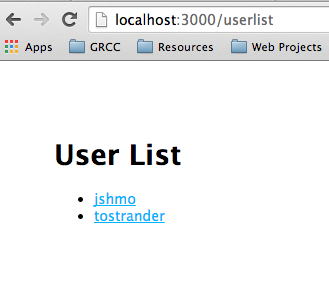
User List

ul

each user in userlist

li

a(href="mailto:#{user.email}")= user.username



1. In your original command window, start the server.   
   Then, navigate to userlist in the browser. Voila!

## Writing to the Database

1. First, we’re going to create a route for an add user form. Open demo/routes/index.js, and add the following code:

/\* GET New User page. \*/

router.get('/newuser', function(req, res) {

res.render('newuser', { title: 'Add New User' });

});

1. Next, we will create a template. Open demo/views/index.jade and save it as newuser.jade. Replace the file contents.

extends layout

block content

h1= title

form#formAddUser(name="adduser",method="post",action="/adduser")

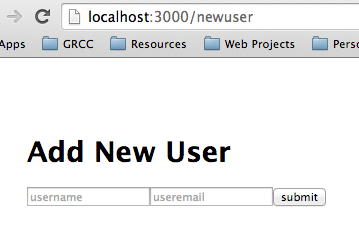
input#inputUserName(type="text", placeholder="username", name="username")

input#inputUserEmail(type="text", placeholder="useremail", name="useremail")

button#btnSubmit(type="submit") submit

This creates a form with method = post and action = adduser. We define two inputs and a button.

1. Restart the node server and navigate to <http://localhost:3000/newuser> to see your form.



1. If you submit the form, you’ll get a 404 error. We need to set up a route to handle POST requests. In demo/routes/index.js, add the following route:

/\* POST to Add User Service \*/   
router.post('/adduser', function(req, res) {   
 // Set our internal DB variable  
 var db = req.db;   
  
 // Get our form values. These rely on the "name" attributes   
 var userName = req.body.username;   
 var userEmail = req.body.useremail;   
  
 // Set our collection  
 var collection = db.get('users');   
  
 // Submit to the DB   
 collection.insert({   
 "username" : userName,   
 "email" : userEmail   
 }, function (err, doc) {   
 if (err) {   
 // If it failed, return error   
 res.send("There was a problem adding the information to the database.");   
 }   
 else {   
 // If it worked, set the header so the address bar doesn't say /adduser   
 res.location("userlist");   
 // And forward to success page   
 res.redirect("userlist");   
 }   
 });   
});

1. Make sure your database server is still running, then stop and restart your node server. Give it a try!

Note: When you’re done, make sure to kill your database server and your node server in their respective terminal windows using Ctrl+C. If you close the terminal window without cancelling, the servers will still be running.

Adapted from <http://cwbuecheler.com/web/tutorials/2013/node-express-mongo/>