COMP3322 Modern Technologies on World Wide Web

NPM & Installation of Node.js Environment

Overview of NPM

Node Package Manager (NPM) is the default package manager for Node.js. It provides two main functionalities:

(1) Provide online repository for Node.js packages/modules, making it easy for programmers to publish and share source code of Node.js libraries. NPM consists of a command line client that interacts with the registry. Over 475,000 packages are available on the main npm registry.

(2) Provide command line utility to simplify installation, updating and uninstallation of Node.js packages, as well as do version management and dependency management of Node.js packages.

NPM is automatically included when Node.js is installed.

More details about npm is at https://www.npmjs.com

Install Node.js Environment

We can install the Node.js runtime environment using the following steps.

Step 1: Go to https://nodejs.org/en/ and download Node.js installation package. Run the installer and install Node.js into a folder at your choice. If you are using Mac OS or Linux, you may just follow the auto-installation program and install Node.js to the default location.

Step 2: Open a terminal, and type in the following command to install the Express framework:

```
npm install  -g express-generator
```

If you are using Mac OS or Linux, you may need to add sudo before “npm”:

```
sudo npm install -g express-generator
```

Step 3: Switch to a working directory at your choice. Create an Express project and name it “test", as follows:

```
express test
```

Then you will see a “test” folder created. Open the “test” folder, you will see that some files have been automatically created by the Express framework, in the following directory structure:

```
app.js
package.json
./routes
```

./views
./bin
./public

Here, **app.js** is the main app file. **package.json** is a JSON file describing the app and its dependencies. We will create router modules in directory ./routes and webpage templates in ./views. We can place static files to be serve by the web app in ./public, and ./bin contains default project files.

NPM can install, in one command, all the dependencies (modules) that the project needs as specified in package.json, as follows:

```
cd test
npm install
```

Once the above installation is done, you should see a ./node_modules folder that contains these modules.

Due to a trademark issue, the Jade template engine has been renamed to Pug. But the Express framework still generates a project that uses the Jade template engine by default. You can follow the steps below to change the template engine from Jade to Pug.

- Install the Pug template engine as follows in the "test" directory:

```
npm install pug
```

- Rename error.jade, index.jade, and layout.jade in ./views folder to error.pug, index.pug, and layout.pug.

Step 4: Open **app.js** in a text editor and check out its content:

- You can see it contains code for loading different modules, specifying routers under "./routes" directory, setting view engines, and setting the middlewares that process all the requests before passing them to middlewares defined in the routers. You should be able to understand the code based on examples in the lecture slides.
- Replace app.set('view engine', 'jade'); by app.set('view engine', 'pug'); to use Pug template engine in the app.
- **app.js** also contains error handling code. Please refer to http://expressjs.com/guide/error-handling.html to get a good idea of the error handling code.
- You can also see this line of code "module.exports = app;" at the end of the file, which exports this *app* as the default module that the Express app will run once started. Then you can start the Express app in the terminal by type the following command in the "test" directory:

if not work, add

```
npm start
```

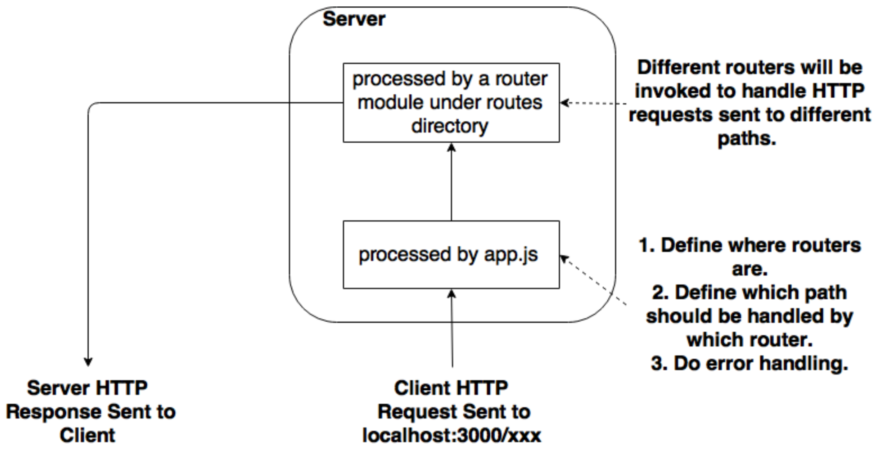
cd test
before this

After running "npm start", the web server is started and listens at the default port 3000.

As introduced during the lectures, an alternative approach to start the server at a given port (e.g., 8081) is to replace "module.exports = app;" by the following code in app.js, and then run the app by typing "node app.js" in the terminal:

```
var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
})
```

Now let's understand how the web server works based on the following figure. When the server receives an HTTP request from a client, the request will first be processed by **app.js**, and then further processed by a router in the "./routes" directory, according to the path it requests.


Server
processed by a router module under routes directory
processed by app.js
Different routers will be invoked to handle HTTP requests sent to different paths.
1. Define where routers are.
2. Define which path should be handled by which router.
3. Do error handling.
Server HTTP Response Sent to Client
Client HTTP Request Sent to localhost:3000/xxx

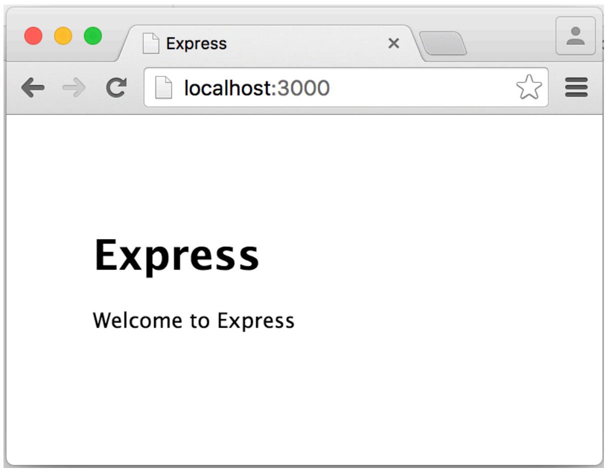
Especially, in **app.js**, you see the following 2 lines, which define 2 router modules. One is **routes**, implemented in ./routes/index.js. The other is **users**, implemented in ./routes/users.js.

```
var index = require('./routes/index');
var users = require('./routes/users');
```

Then the following 2 lines define which router module should handle which path. The **routes** module will handle HTTP requests sent to "http://localhost:3000/" while **users** module handles HTTP requests sent to "http://localhost:3000/users".

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

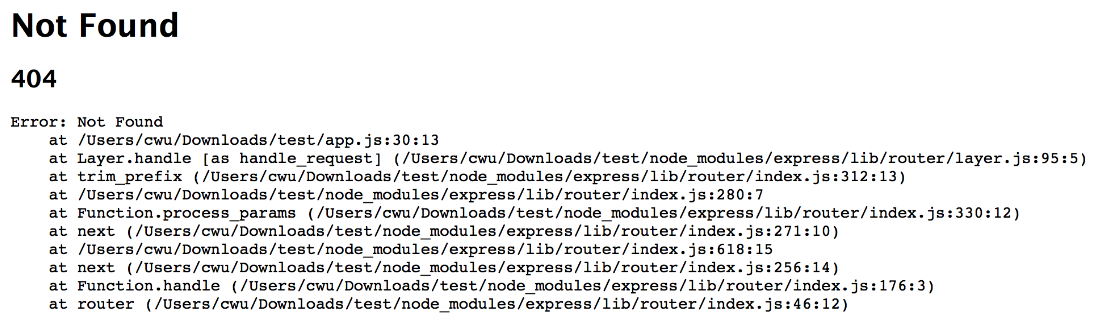
Step 5: Launch a web browser and check out the default page at http://localhost:3000. You should see a page as shown in the following figure. This default page is rendered by the middleware in "./routes/index.js", using **index.pug** under the "./views" directory.



Express

Welcome to Express

Next try to access the link http://localhost:3000/incorrectPath in your browser. You should see something like this:



Not Found

404

```
Error: Not Found
    at /Users/cwu/Downloads/test/app.js:30:13
    at Layer.handle [as handle_request] (/Users/cwu/Downloads/test/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/Users/cwu/Downloads/test/node_modules/express/lib/router/index.js:312:13)
    at /Users/cwu/Downloads/test/node_modules/express/lib/router/index.js:280:7
    at Function.process_params (/Users/cwu/Downloads/test/node_modules/express/lib/router/index.js:330:12)
    at next (/Users/cwu/Downloads/test/node_modules/express/lib/router/index.js:271:10)
    at /Users/cwu/Downloads/test/node_modules/express/lib/router/index.js:618:15
    at next (/Users/cwu/Downloads/test/node_modules/express/lib/router/index.js:256:14)
    at Function.handle (/Users/cwu/Downloads/test/node_modules/express/lib/router/index.js:176:3)
    at router (/Users/cwu/Downloads/test/node_modules/express/lib/router/index.js:46:12)
```

This is handled by the error handling code in **app.js**, as follows:

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});
```

Run the Examples in Lecture Slides

We have provided source code of examples from the lecture slides in examples.zip. Please try them out using the Node.js environment you have just installed, to acquire a good understanding of the working of Node.js and Express.js.

Example 1: This is the simple Node.js web serve app created by main.js. Go to example1 directory and run the server, as follows:

before you use "cd", you should first put the example1 file in the /users/KHH directory.

```
cd exampe1    this should be example1, but not exampe1
node main.js
```

Then the server is up running at port 8081 on your computer. Launch a browser and access the URL http://127.0.0.1:8081 to check the page out.

Note: you should always use control+C to kill an already running server app before you start the server again after making modifications), when you run the following examples.

Example 2: This is the simple Express.js web server app created by app.js. You can reuse the Express project directory you created just now, i.e., "test". Copy app.js in our given example2 folder to the "test" project directory, overwriting the default app.js. Then in the project directory, run the app as follows:

```
node app.js
```

Then the server is up running at port 8081 on your computer. Launch a browser and access the URL http://127.0.0.1:8081 to check the page out.

Example 3: This is the Express.js web server app (app.js) with a mountable router module (birds.js). You can reuse the "test" project you created. Copy app.js and birds.js in our given example3 folder to this project directory, overwriting the previous app.js. Then in the project directory, run the app by command "node app.js".

Then the server is up running at port 8081 on your computer. Launch a browser and access the URL http://127.0.0.1:8081/birds/ and http://127.0.0.1:8081/birds/about to check the pages out.

Example 4: This is the Express.js web server app to serve static files. Copy app.js in our given example4 folder to your project directory, overwriting the previous app.js and removing the previous birds.js. Copy some static files into the public directory, e.g., copy puppy.jpg into public/images. Then in the project directory, run the app by command "node app.js".

Then the server is up running at port 8081 on your computer. Launch a browser and access the URL http://127.0.0.1:8081/images/puppy.jpg to check the image out.

Example 5: This is the Express.js web server app to serve static files and handle form data sent by HTTP GET requests. Copy app.js and index.html in our given example5 folder to your project directory, overwriting the previous app.js. Then in the project directory, run the app by command "node app.js".

Then the server is up running at port 8081 on your computer. Launch a browser and access the URL http://127.0.0.1:8081/index.html to check the page out, fill in the form and then click submit to see the generated page.

Example 6: This is the Express.js web server app to serve static files and handle form data sent by HTTP POST requests. Copy app.js and index.html in our given example6 folder to your project directory, overwriting the previous app.js and index.html. Then in the project directory, run the app by command "node app.js".

Then the server is up running at port 8081 on your computer. Launch a browser and access the URL http://127.0.0.1:8081/index.html to check the page out, fill in the form and then click submit to see the generated page.

Example 7: This is the Express.js web server app with cookie management. Copy app.js in and index.html in the given example7 folder to your project folder, overwriting the previous app.js and index.html. Then in the project directory, run the app by command "node app.js".

Then the server is up running at port 8081 on your computer. Launch a browser and access the URL http://127.0.0.1:8081/. You should see a page showing "This is the index page". Then refresh the page and you should see a page showing "Click to forget". If you click the "forget" link, you should be directed back to the web page showing "This is the index page".

Example 8: This is the Express.js web server app with session management. Copy app.js in our given example8 folder to your project folder, overwriting the previous app.js. Keep index.html from Example 7 in your project directory.

The "express-session" module has not been installed in your project directory, since its dependency is not specified in the default package.json. Run the followig command to install the module:

```
npm install express-session
```

Then in the project directory, run the app by command "node app.js" (you can ignore the warnings on the console). The server is up running at port 8081 on your computer. Launch a browser and access the URL http://127.0.0.1:8081/ to check the pages out. You should see similar web pages as in Example 7, except that session is used instead of cookie for managing user states.

Example 9: This is the Express.js web server app with Pug template. Copy app.js in our given example9 folder to your project folder, overwriting the previous app.js. Remove index.html from your project directory. Copy index.js to routes folder in your project directory, replacing the default index.js there. Copy index.pug and helloworld.pug to views folder in your project directory, replacing the default template file. In the project directory, run the app by command "node app.js".

Then the server is up running at port 8081 on your computer. Launch a browser and access the URLs http://127.0.0.1:8081/ and http://127.0.0.1:8081/helloworld to check out the generated pages.