COMP3322 Modern Technologies on World Wide Web

NPM & Express Generator

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 are at https://www.npmjs.com

Express Generator

In the lecture, we use npm init to initialize a project and then install express to our project. There is another way to build an express project – use the express generator. Our NodeJS docker has already installed with the express-generator module.

Step 1: Switch to a working directory at your choice. Create an Express project that uses the ***Pug template engine*** and name the project "test", as follows:

```
express test --view=pug
```

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:

```
test
├───app.js
├───bin
│   └───www
├───package.json
├───public
│   ├───images
│   ├───javascripts
│   └───stylesheets
│       └───style.css
├───routes
│   ├───index.js
│   └───users.js
└───views
    ├───error.pug
    ├───index.pug
```

```
        └──layout.pug
```

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 many modules (including express and pug).

Step 2: 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.

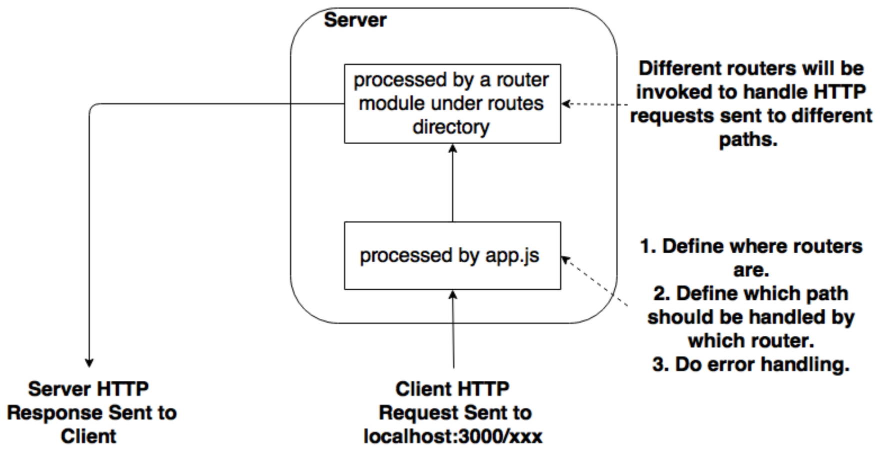
app.js also contains error handling code. Please refer to http://expressjs.com/guide/error-handling.html to get a good idea of the error handling code.

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:

```
npm start
```

After running "npm start", the web server is started and listens at the default port 3000. **To use another port number, you can edit the file 'www' in the bin folder and change the port number to another number**.

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
Different routers will be invoked to handle HTTP requests sent to different paths.
processed by app.js
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

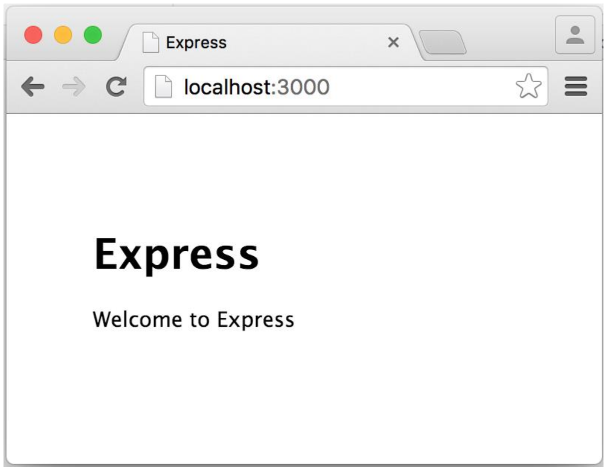
Especially, in app.js, you see the following 2 lines, which define 2 router modules. One is routes, implemented in ./routes/index.js. The other is users, implemented in ./routes/users.js.

```
var indexRouter = require('./routes/index');
var usersRouter = 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('/', indexRouter);
app.use('/users', usersRouter);
```

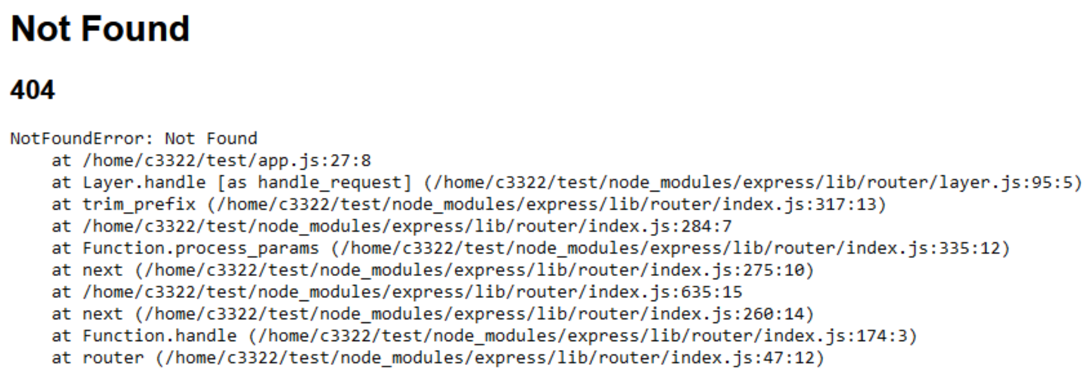
Step 3: Launch a web browser and check out the default page at http://localhost:3000. You should see a page as shown in the following figure. This default page is rendered by the middleware in "./routes/index.js", using index.pug under the "./views" directory.



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

```
NotFoundError: Not Found
    at /home/c3322/test/app.js:27:8
    at Layer.handle [as handle_request] (/home/c3322/test/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/home/c3322/test/node_modules/express/lib/router/index.js:317:13)
    at /home/c3322/test/node_modules/express/lib/router/index.js:284:7
    at Function.process_params (/home/c3322/test/node_modules/express/lib/router/index.js:335:12)
    at next (/home/c3322/test/node_modules/express/lib/router/index.js:275:10)
    at /home/c3322/test/node_modules/express/lib/router/index.js:635:15
    at next (/home/c3322/test/node_modules/express/lib/router/index.js:260:14)
    at Function.handle (/home/c3322/test/node_modules/express/lib/router/index.js:174:3)
    at router (/home/c3322/test/node_modules/express/lib/router/index.js:47:12)
```

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) {
  next(createError(404));
});

// 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');
});
```