BOSTON UNIVERSITY

Department of Computer Science

CS411 Node and Express Walkthrough

@perrydBUCS

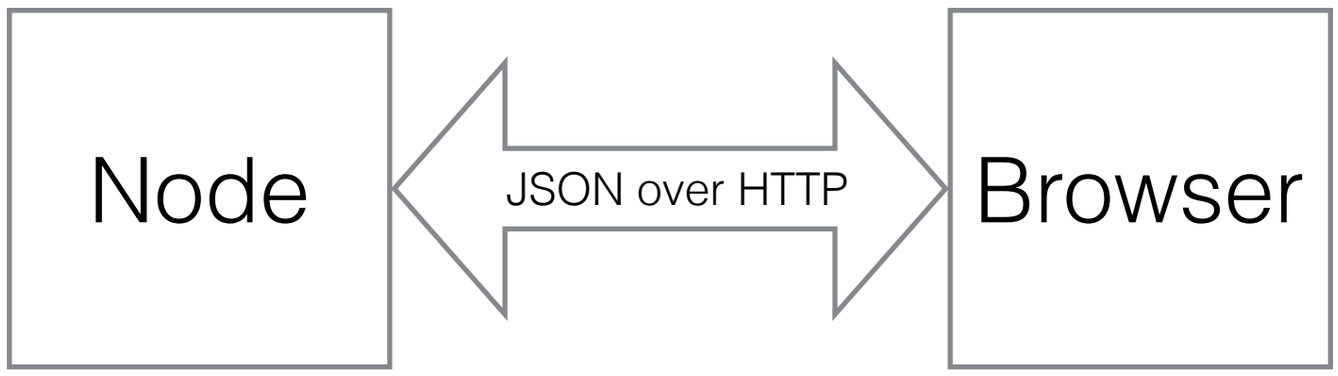
Node.js is

- About five years old
- Web server
- Server-side Javascript engine (using Google V8)
- HTML router using Express
- Application development framework
- Supported by modules and the node package manager (npm)
- On the ‘hot list’ for jobs

App architecture

- Most Node applications are separated into front end and back end
- Node handles the back end by providing a RESTful API
- Node is the **model** in this architecture
- The front end (typically a browser) acts as **controller and view**, and calls the API on the back end to do its work

- This lets us work on the front and back end independently since they are loosely coupled
- We use JSON as the transport language for the API



npm

- Functionality in Node is provided by modules
- A package manager, npm, makes it easy to find and install functionality that you need
- npmjs.org is the site to explore, and you can install packages from the command line
- npm handles dependencies, so if, for example, express requires range-flatten.js, npm will grab and install it
- The front-end equivalent is yarn

Installing Node.js

- Download and install Node.js from nodejs.org
- Open a terminal window
- Install express-generator with -g option (globally)
  - `sudo npm install -g express-generator`

Create fresh Node / Express project

- Open up a terminal
- cd to the directory in which you want your project
- *express proj-name*
- *cd proj-name*
- *npm install*
- Test the app: *npm start*
- Server is at http://localhost:3000

mongoDB

- If you need to install mongoDB:
  - *brew install mongodb*
- Start mongo in daemon mode:
  - *mongod&*

Three key concepts

- Routing
- Middleware
- Callbacks

Routing

- Routes are the way Node responds to HTTP requests
- Node keeps a list of URL paths and methods and provides a way to respond to each
- Methods are GET, PUT, POST, DELETE, etc
- The API is constructed with methods and URLs:

```
router.get('/', function(req, res, next) {
 res.render('index', { title: 'Express' });
});
```

Middleware

- Each time the server receives a request, it runs a series of functions on the request in a specific order
- This middleware allows us to do things like logging, authorization, validation and so on on each request
- Middleware is installed using app.use() and is run in the order that it appears in the file
- Each middleware function takes three arguments: HTTP request and response object, and a *next()* function to run

Callbacks

- Most functions in Express take another function as the final argument
- The argument is used to implement a callback…when the function we are using is finished, it passes control to the last-argument (callback) function
- The callback is typically implemented as an anonymous function

```
router.get('/find', function (req, res, next) {
  people.find({}, function (err, results) {
    return res.json(results);
  });
});
```

- All middleware functions are passed the HTTP request and response objects
  - request has the original HTTP data, including headers and body
  - response is what gets sent back to the browser at the end of the function
- Since they are middleware they also take a *next* parameter which will fire when the current function returns…that's how middleware is chained together
- Most built-in middleware modules call the next function by default, but if you write your own it must call next()

Return values from most modules

- Most modules will return something on each function
- The first return value will be an error object
  - If the error object is null, then all is well
  - If the error object is not null, it contains info on what went wrong
  - Note that this doesn't happen magically…the error object contains whatever the coder of the module decided to put into it
- Remaining params will be return values from the module
  - Each is different, you'll need to look up the documentation of what is returned

Let’s do it

- We’ll create a new Node/Express project
- The app will hit an API that retrieves currency exchange rates

  GET - calculate 100USD in EUR (default)
  POST - calculate a given USD amount in EUR

- We’ll add a small logging component to demonstrate middleware
- We’ll use Postman for testing
- There’s no UI yet…we’ll look at Angular on the front end in a few weeks