Catbook Workshop: Intro to NodeJS

Monde Duinkharjav
What is NodeJS?
NodeJS is...

A Javascript RUNTIME ENGINE

NOT a framework

NOT Javascript nor a JS package

It is a method for running your code in Javascript. Let’s see how…

In your terminal run the command
$ nodejs

> node (for Windows users)
> console.log("Hello World");
Hello World
Pop Quiz: Where else can you find a javascript runtime engine?
Programming Language Libraries

Programming languages have many libraries.

E.g. numpy, scipy for python.

In python, we have pip. This manages your python packages.

In javascript, we have...
Node Package Manager (NPM)
Initializing a Javascript Project using NPM

$ npm init

This command creates package.json. This file keeps track of info about the project

```json
{
  "name": "Catbook",
  "version": "1.0.0",
  "description": "Facebook for cats",
  "main": "app.js",
  "author": "6.148 Staff",
  "license": "MIT"
}
```
Let’s create a web server using the http package

Create a file called app.js. And in this file, type:

```javascript
1 console.log("Hello World");
```

Now, running the command you will get:

```
$ nodejs app.js
Hello World
```
Let’s create a web server using the http package

$ npm install http --save

(parallel to pip install)
This command will do two things:

1. Add the http package
   DEPENDENCY to your package.json
2. Download the http library in under
   the directory node_modules

WARNING: this installs http only for the current project

```json
{
  "name": "Catbook",
  "version": "1.0.0",
  "description": "Facebook for cats",
  "main": "app.js",
  "author": "6.148 Staff",
  "license": "MIT",
  "dependencies": {
    "http": "*"
  }
}
```
IMPORTANT TANGENT

Make a file called `.gitignore` in your project’s directory.

And write inside it: `node_modules/`

Why? You don’t want to push thousands of lines of code from libraries into your repository. Instead, you make git ignore the `node_modules` directory.

When you later clone your project code, you always run

```
$ npm install
```

And reinstall `node_modules` to your machine.
Let’s import the http package into our project

In app.js

(equivalent to import numpy for python)

This loads the http library into our project. Let’s use this package and start a webserver:

You can run the app now with:

$ nodejs app.js
If you’re lost:
Go to go.6148.io/workshop2 and clone
`git checkout step1`
function(req, res) {}

1. req stands for request, res stands for response
2. We’re passing in a function to these methods:
   Whenever a request is made, this function is invoked.
   The argument res is mutated by this function.
3. This type of function pops up a LOT in web server request handling
Let’s test our web server out

Open your web browser: http://localhost:3000

Okay, we’re done! We have a web app. Thank you for coming. You’re done…
Let’s test our web server out

Real applications are many orders of magnitude more complex so the code base can get out of hand really quickly.

Introducing: Express

A Javascript library that provides a web server architecture.

Rupayan’s Take on Express: Organize my web server.

Let’s install it... hold up. Let’s do some housekeeping first.
Housekeeping

1. Create a directory called src and public
2. Move app.js into src
3. From here on we’ll use a wrapper to nodejs: nodemon
   a. Run $ npm install nodemon --save-dev
   b. Add npm start script to package.json

```json
{
  "name": "Catbook",
  "version": "1.0.0",
  "description": "Facebook for cats",
  "main": "app.js",
  "author": "6.148 Staff",
  "license": "MIT",
  "scripts": {
    "start": "nodemon src/app.js",
    "dependencies": {
      "express": "^4.16.1",
    },
    "devDependencies": {
      "nodemon": "^1.12.1"
    }
  }
}
```
What happened to our project?

1. Server logic code is now in `src`. Static files accessible for all clients in `public`.
2. Nodemon detects when you change the files in your app and restarts your app.
3. Dev Dependencies are the libraries you use for development. Good to have this distinction for development vs actual application logic.
4. `$ npm start` is an ALIAS to typing `nodemon src/app.js` which is a good abstraction to make.
Using express to send a Hello World message

```javascript
const http = require('http');
const express = require('express');

const app = express();

app.get('/', function(req, res) {
  res.send('Hello World');
});

const port = 3000;
const server = http.Server(app);
server.listen(port, function() {
  console.log('Server running on port: ' + port);
});
```
Exercise: Create a new endpoint

Aside: endpoint - a reference to a service a web application provides. E.g. ‘/’ is the “root endpoint”. We can have a ‘/u/profile’ endpoint that can be reached by typing http://<webapp url>/u/profile

Exercise: Create a new endpoint called ‘/u/profile’ that sends the message ‘My Profile’.

If you’re behind, you can run to hop in right before you need to do this exercise

```
git reset --hard
git checkout step2
```
Now let’s send our .html files instead of plain text

1. Create a directory called ‘/src/views’
2. Change `res.send` to `res.sendFile`

```javascript
text(app.get('/', function(req, res) {
    res.sendFile('index.html', { root: 'src/views' });
});
```
Now let’s send our .html files instead of plain text

Oops… when we request the endpoint, the css and js files don’t work properly…

Let’s fix that:

1. Add line 6 to your code
2. Move the js and css folders into public
3. Change the references in the html

```javascript
const http = require('http');
const express = require('express');

const app = express();

app.use('/static', express.static('public'));

app.get('/', function(req, res) {
    res.sendFile('index.html', { root: 'src/views' });
});

const port = 3000;
const server = http.Server(app);
server.listen(port, function() {
    console.log('Server running on port: ' + port);
});
```
Exercise: Render the ‘/u/profile’ endpoint

- ‘/u/profile’ endpoint should render the profile.html page we created on Tuesday’s lecture.
Okay moving on... more Housekeeping

What happens if we try to reach the endpoint

'/aaronripsersucks' or '/dannysrealnameisdanielwinstontang'
Okay moving on... more Housekeeping

What happens if we try to reach the endpoint

'/aaronripsersucks' or
'/dannysrealnameisdanielwinstontang'

404!!!
Okay moving on... more Housekeeping

We should also take care of 500 Internal Errors.

Not really important how this works exactly... tbh i don’t exactly know how this works myself.
Working with several files

As projects get larger, one app.js file will be too much to deal with. Let’s demonstrate how this is done by moving our “view” endpoints somewhere else.

1. Create a directory called ‘src/routes’
2. Inside, make a file called views.js

```javascript
// dependencies
const express = require('express');
const router = express.Router();

// public endpoints
router.get('/', function(req, res, next) {
    res.sendFile('index.html', { root: 'src/views' });
});

router.get('/u/profile', function(req, res) {
    res.sendFile('profile.html', { root: 'src/views' });
});

module.exports = router;
```
module.exports = router

Remember?

const http = require('http');

This is importing http. Then, you export a package by typing:

module.exports = ...

We exported router just now, we import it back in our app.js by typing:

```javascript
// local dependencies
const views = require('./routes/views');
```
Final results: go.6148.io/workshop3