Webservers

A program that

- accepts incoming web requests
- sends outgoing web response

Express

http://expressjs.com/

- A framework to help write webservers in NodeJS.
- One of many
    - A commonly used one
- We use it to see what webservers do

Create a new package

1. Make a dir for your new project: `express-test/`
2. Inside `express-test/` (`cd express-test`)
    - run `npm init -y`
    - Creates a new package (your package)
        - Using default answers to setup questions
        - Doesn't do anything remote
            - Only local changes
3. a `package.json` was created
    - take a look

Installing Dependencies

Our new package can track **dependencies**

- Still inside `express-test/`
- run `npm install express`
- Not with `sudo`, not as admin

This is a **local install**

- inside this package

Because we are "in" a package, `package.json` updates

- and a `node_modules/` was created

How Generic is this info?

- package.json is true for all Node-based projects
    - but is Node-specific
- Some form of dependency management true for most languages
    - Ability to know what versions installed
    - Ability to fetch and install/re-install
    - Ability to update

JSON

- JavaScript Object Notation
- A text format that easily translates to Javascript
- Note it is TEXT, not JS

JSON Can Not...

Cannot:

- Have comments
- Store functions/methods
- Store construction/"class" information

Simple and durable, but highly limited to data

JSON Formatting

More strict and limited than JS formatting:

- Any quoting is only double-quoting
  - JS accepts single/double/backtick
- All object keys are explicitly quoted
  - JS does not require quoted keys unless special characters
- No trailing commas in objects/arrays
  - JS (ES6+) allows trailing commas
- Whitespace still irrelevant

package.json

A JSON file that every npm-using package has

- It contains information about the package
    - including dependencies
- Use even for private packages

What does package.json give us?

- delete `node_modules/` and contents
- run `npm install` (in `express-test/`)
    - `npm install` vs `npm install express`
    - Recreates `node_modules/`
    - This is how we will test your work
    - This is how teammates stay in sync
    - This is how users use libraries
    - Often how the servers you deploy to work
    - Allows "delete and reinstall"
        - "turn it off and back on again" of node

package.json parts include

https://docs.npmjs.com/files/package.json

- package name
- version (in semver)
- dependencies list
    - lists version or minimum version
    - devDependencies (for those working on the package itself)
- Author/repo info
- License
- Scripts

package.json scripts

Lists shell (command-line) commands to run with

- `npm run SCRIPTNAME`
- e.g. script of `"greet": "echo Hello"`
    - prints "Hello" when you run `npm run greet`
- Used to collect commands for users
- Used to build your package (more later)

A few pre-defined script names don't require "run"

- e.g. `npm test` is the same as `npm run test`

Written well, scripts work on many operating systems

Versioning

There is no universal truth to version numbers

- May be marketing (MS Word)
- May be date-based (Minecraft betas)
- May be dev vs prod (Linux kernels)
- May be weird (TeX and MetaFont)

Hard to reliably parse, compare, understand

SemVer is an attempt at meaningful versions

- Not just JS or web, all software

Semver - Semantic Versioning

https://semver.org/

- X.Y.Z - three numbers
- ".x" means "any"
    - so 2.x means 2.(whatever)
- NOT like decimal
    - 1.1 is NOT 1.10
    - 1.10.0 after 1.9.x
        - 2.0.0 "later" than 1.9 and 1.10

Semver parts

MAJOR.MINOR.PATCH

- MAJOR version is an API-breaking change
    - or likely breaking change
- MINOR version is a new feature
    - no breaking changes
- PATCH version is a bugfix
    - no breaking changes
- 0.x.x means nothing is stable

Additional rules for betas and release candidates

package.json dependencies

Runing `npm install` in a dir with a package.json

- npm will install all of the dependencies
    - (recursively) into `node_modules/`

`node_modules/` should not be put into version control

When you `require()` a file without a path:

- npm looks in `node_modules/`

Dependencies Versions

What does the `dependencies` object mean?

- `x.y.z` = exact version
- `^x.y.z` = latest of this major version
- `~x.y.z` = latest of this minor version

Those versions are what `npm install` installs

- creates `package-lock.json`
  - records EXACT versions installed
  - USUALLY should be put in source control (git)
  - Used during deployment
    - recreate exact versions tested

Why so many dependencies

Is this SO MUCH CODE?

- Not really
- node libs tend to small and numerous
- Some other languages go for large and few

Dependencies (either style) ARE a security concern

- But often a necessary one
- Who to trust?
- How to learn about vulnerabilities?
- How to stay up-to-date on versions?

Create your static assets

Inside `express-test`, create a `public` directory.

- This will hold **static** files and assets
- This will be the webserver root for static assets

Create an `index.html`

- inside `public/`
- that says "Hello World"

Basic Express Webserver

```
const express = require('express');
const app = express();

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

app.listen(3000, () => {
  console.log('listening on http://localhost:3000');
});
```

Confirm the static assets

- run `node server.js`
- view `http://localhost:3000` in browser
    - Why 3000? Random but common dev port.
    - Ports below 1024 require admin permissions
    - Common web dev ports:
        - 3000, 4000, 5000, 9000
        - 8000, 8080, 8443, 8888
    - Usually only 80/443 for real internet access
        - these dev ports only for local access
            - occassionally intranet/VPN access

Document Root and Static Assets

The `public/` directory is our **document root**

- Files in it are viewable using our server
- Links to these assets will NOT include `public/`
    - Common source of confusion

If we have:

```
public/
public/index.html
public/css/styles.css
```

- `<link rel="stylesheet" href="/css/styles.css">`
    - relative path also works
    - but no `public/` in `href` path

`express.static`

```
app.use(express.static(DOCUMENT_ROOT));
```

- Defines directory to use as **document root**
- Will try to match paths and files from requests
  - To paths and files in that directory

Adding a dynamic asset

server.js before `app.listen`:

```
app.get('/dynamic.html', (request, response) => {
  response.send('This is not an actual file');
});
```

public/index.html:

```
<a href="dynamic.html">See a Dynamic page</a>
```

Restart `node server.js`

Confirm you can follow the link to the dynamic page

Why Restart?

Static asset changes **don't** require the server to restart

Dynamic asset changes **do** require the server to restart

Why?

Express Routes

We give Express a collection of **routes** and **callbacks**

If request matches the route

- Call callback
- Pass a **request** and **response**
- Callback will decide response

Server loops through all routes for each request

Summary - Webservers

Webservers are programs

- listen for incoming web request
- sent a web response

Response may be a **static asset** (file)

- May be a **dynamic asset** (generated)

Response is always based on request

- Path may be used as file-system path
- Path may be used as data
- Path may be ignored!

Summary - Express

We run a server using `express` library

- it **listens** to a **port**
- Each request passes through a series of checks
    - Does the request match these terms?
    - If so, call this callback
        - Callback will decide to send response
        - ...and what to send
        - ...and if to pass on to more checks

This is a **Chain of Responsibility** pattern

Summary - Package.json

- `npm init` creates a **package.json**
- package.json has info on the package
- including the dependency list
- `npm install` will install the dependencies
    - needs to run inside the package directory!
- `package-lock.json` normally committed to repos
    - `node_modules/` is NOT in repos

Summary - JSON

JSON is a **text format**

- Looks like JS, is not JS
- Easily translated to/from JS
- Only holds "data", nothing runnable
  - no functions
  - no classes
- Has strict formatting requirements
  - all quoting is double-quotes
  - object keys are quoted
  - no trailing commas
  - no comments