Express

Express is framework to help write webservers in NodeJS.

It is one of many, but is a commonly used one

Starting

1. Make a project directory
2. **Inside** project directory, run `npm init`
    - Accept defaults for each prompt
    - Creates `package.json`

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 if you aren't sharing your package

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

Semver - Semantic Versioning

https://semver.org/

- Not just JS or web-related
- MAJOR.MINOR.PATCH - three numbers
- NOT like decimal
    - ".x" indicates "any", so 2.x is any version 2
    - 1.10.0 is > 1.9.x, but 2.0.0 is "later" than both
- MAJOR version is an API-breaking change
- MINOR version is a new feature
- PATCH version is a bugfix, no required changes
- 0.x.x means nothing is stable

package.json scripts

Lists any shell (command-line) commands that will run with `npm run SCRIPTNAME`

- e.g. script key of `"greet": "echo Hello"` will echo "Hello" when you run `npm run greet`

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

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

Scripts are very handy for collecting commands for users or building your package

With some effort scripts can run on many operating systems

Install Express

Inside your project directory, run:

- `npm install express`
    - Note: not global, local install
    - **Not** `npx`

Look at your directory

- See there is now a `node_modules/` directory
- See there is now a `package-lock.json` file

Node Modules

This directory is managed by `npm`

- Holds all the dependencies of your package
- ...including their dependencies

You only installed `express`, but it installed dozens!

- These are needed for express
- Many packages are very, VERY small
- Many packages != bloat!

Package.json has changed!

There is now a `dependencies` list!

- lists `express`
- lists a version...with a `^`?

package.json dependencies

`node_modules/` should NOT be put into version control

When you `require()` a file without a path, npm will look in `node_modules/`

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

Installing dependencies

Run `npm install` where a `package.json` is present

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

The package-lock.json file

- `package.json` allows a range of dependency versions
- `package-lock.json` records EXACT versions
    - Used in deployment to make servers match the testing
- Internally, probably commit the package-lock file
- Externally, no one uses it
- For this course, I don't care
    - Just know what the file is for

Create your static assets

Inside your project directory, create a `public` directory.

- This will hold **static** files and assets
- This will be the **document** root for static assets
- It can have subdirectories (like `css/` or `images/`)

Basic Express Webserver

Create a `server.js`

- **Not** in `public/`
- We pick `server.js`, that name is not required

```
const express = require('express');
const app = express();
const PORT = 3000;

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

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

Confirm the static assets

- Create an `index.html` in the `public/` directory
- Run `node server.js`
- View `http://localhost:3000` in browser
    - Note: **3000**, not 5000 like we used with `serve`
    - Why 3000? Just to distinguish from serve. No extra meaning.
    - We use non-standard ports because standard ports require root/admin

Adding a dynamic asset

server.js before `app.listen`:

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

Add to **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?

Changed static assets **don't** require a server restart

Changed dynamic assets **do** require a server restart

Why?