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 directory for your new project: `express-test`
2. Inside `express-test/`, run `npm init -y`
    - Tells npm that this is a new package (your package)
    - Which includes the ability to install local packages/modules
3. install express: `npm install express`
    - LOCAL, not global
    - NOT with `sudo`
4. Examine `package.json` and `node_modules/`

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 intending to share 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
- X.Y.Z - three numbers
- NOT like decimal
    - ".x" indicates "any", so 2.x means 2.(whatever)
    - 1.10.0 follows 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 without breaking anything
- 0.x.x means nothing is stable

package.json dependencies

When you run `npm install` where a package.json is present, 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 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

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

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
    - Note: **3000**, not 5000 like we used with `serve`
    - Why 3000? Just to distinguish from serve. No extra meaning.

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?

Changes to the static assets **don't** require the server to restart

Changes to the dynamic assets **do** require the server to restart

Why?