Node and Express

- Writing a webserver to serves a web application
- Many items
- We'll learn each as we go

NodeJS

`node` - JS Engine for JS outside browser

- Not just as a server
- `deno` and `bun` are two similar engines

NPM

- Create an empty work folder: `express-test`
    - Not within your repository

Inside `express-test/`:

```
npm init -y
```

- Creates a `package.json` file
    - More on that shortly

What is NPM?

Node Package Manager (npm)

- A "Registry" of libraries and tools using node
- Command to interact with that registry
- Manages your `package.json`
- Manages/Installs related libraries and tools

Similar (but not identical) to

- nuget (C#)
- maven (Java)
- pip (Python)
- gems (Ruby)

What does `npm init -y` do?

`npm init` makes folder a **package**

- Creates a `package.json` for current folder
- Asks questions to fill in minimal data
- `npm init -y` is `npm init` with default answers
    - Note: Requires `kebab-case` folder name

So far only changes are on your system

- Nothing published elsewhere
- May never publish to `npm` registry
- Teams will use package internally though

Why do we want a package.json?

Defines/allows project to

- Libraries required to use (**dependencies**)
- Libraries required to change (**devDependencies**)
- Any related commands to run

Improve sharing code

- Here: You provide package, TA/I install and use
- Team: Devs each install and run to **develop**
- Team: Devs each update/share code and package
- Team: Server(s) install and run to **deploy**

Express

`express` is a **framework** to build webservers

- Creates app type
- Has rules/conventions code must follow

Express is a fairly **unopinionated** framework

- More freedom in how you use it
- May involve more effort/more repetition to use
    - "**boilerplate**"

Express syntax closely matches HTTP expectations

Installing Express

In `express-test` folder (where `package.json` is):

- `npm install express`
- Does a **local install** of express
    - Installs just to this package
- We get `node_modules` folder
- We get `package-lock.json` file
- `package.json` file is updated

JSON

- JavaScript Object Notation
- **text format** listing structure data
    - JS-like syntax
    - Easily translates to Javascript
    - Can also translate to other languages
        - Java, C#, Python, etc
- **text**, even though it looks like JS
    - Structured Data like XML, YAML

Why is JSON?

- A simple representation of data structures
    - Serialization
- Contains no executable code
- "Dumb", in a good way
- Servers love to exchange data
    - In diverse structures
- XML is powerful and complicated
    - Involves lots of security issues
- JSON is limited and simple
    - But solves many common needs

JSON Can...

- Represent Numbers, Strings, Booleans, `null`
- Represent "plain" Objects
- Represent Arrays

JSON Cannot...

- Have Comments
- Represent functions/methods
- Represent construction/"class" information
- Represent `undefined`
- Represent `Map()` or `Set()` data
    - Why I've limited these

Simple and durable, but **highly limited** to data

JSON Formatting

More strict and limited to JS formatting

- String quoting must be **double-quotes**
  - JS accepts single/double/backtick
- All **object keys must be quoted strings**
  - JS does not require most keys be quoted
- No trailing commas in objects/arrays
  - JS (ES6+) allows trailing commans

Whitespace is irrelevant in JSON and JS

Example JSON

```
{
    "cat": {
        "name": "Jorts",
        "age": 3,
        "buttered": false,
        "toys": [ "mousie", "laser pointer" ]
    },
    "dog": null
}
```

- Only allowed types
- Strings are double-quoted
- Object keys are double-quoted
- No trailing commas
- Looks like JS, is actually text

Converting to/from JSON

- `JSON.stringify()`
  - Converts passed value to a JSON String
  - Invalid values **silently** dropped or made `null`
    - Ex: `Map()`/`Set()` are quietly lost
- `JSON.parse()`
  - Converts passed string to JS value
  - Throws errors if it can't convert

Where is JSON used?

- Many places, even when JS isn't used
    - "Safe", common data exchange format
    - Used with many languages
    - Common in web services (more later)

`package.json` file updated

- New section `dependencies`
- Lists **express**
- With a funky series of numbers
    - The **semver version number**

What is a **dependency**?

- A needed package to run this package

Running `npm install SOMEPACKAGE`

- Installs that package locally into `node_modules`
- Adds that package version to **dependencies**
    - In `package.json`

Running `npm install` (no package listed)

- Locally installs all dependencies for *this* package
- Installs into `node_modules` folder
- You can delete `node_modules` and run `npm install`
    - The "turn it off and back on again" of node

Some core parts of `package.json`

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

- Package `name` (**kebab-case**)
- `version` (in **semver** notation)
- `dependencies` list (using **semver**)
- `devDependencies` list (more later)
    - For those modifying the package
- Author/repo info
- `license` (permission to use and restrictions)
- `scripts`

Code has "versions"

No universal truth of version numbers

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

Hard to reliably parse, compare, understand

SemVer an attempt at meaningful versions

- Not just JS, not just web, all software
- Even used for some non-software (docs!)

SemVer - Semantic Versionsing

https://semver.org/

- MAJOR.MINOR.PATCH - three separate numbers
- ".x" means "any"
    - Ex: `2.1.1` and `2.1.5` are both part of `2.1.x`
- NOT decimal system
    - `1.1.x` is NOT `1.10.x`
    - `1.10.x` is after `1.9.x`
        - `2.0.0` is "later" than both `1.9.x` and `1.10.x`

Semver Major, Minor, and Patch

- Raise **Major version** when **breaking** changes
  - At least a *potential* breaking change
  - Raising Major version resets Minor and Patch to `.0.0`
- Raise **Minor version** when adding features
  - But using existing features requires no changes
  - Raising Minor version resets Patch version to `.0`
- Raise **Patch version** when making bug/security fixes
  - But nothing else added/changed
  - No changes required by code using this package
- `0.x.x` means *any change* may be breaking
- Additional syntax for **betas**, **release candidates**, etc

`package.json` Dependencies using SemVer

- `x.y.z` - Will only install exact listed version
- `^x.y.z` - This or latest of this **major version**
    - Only updates **minor** or **patch** versions
    - Ex: `^4.12.0` will install `4.12.1` or `4.13.0`
        - Will not install `4.11.0` or `5.0.0`
- `~x.y.z` - This or latest of this **minor version**
    - Only updates **patch** versions
    - Ex: `~4.12.0` will install `4.12.0` or `4.12.2`
        - Will not install `4.11.0` or `4.13.0` or `5.0.0`

`package-lock.json`

- Records EXACT version installed
- And from where
- Should be put in source control
- Updated when you install new versions

Used during **testing** and **deployment**

- Recreate exact versions used in development