Udacity- Full Stack JavaScript Nanodegree Notes

January 05, 2024 - N/A

Courtney Downs

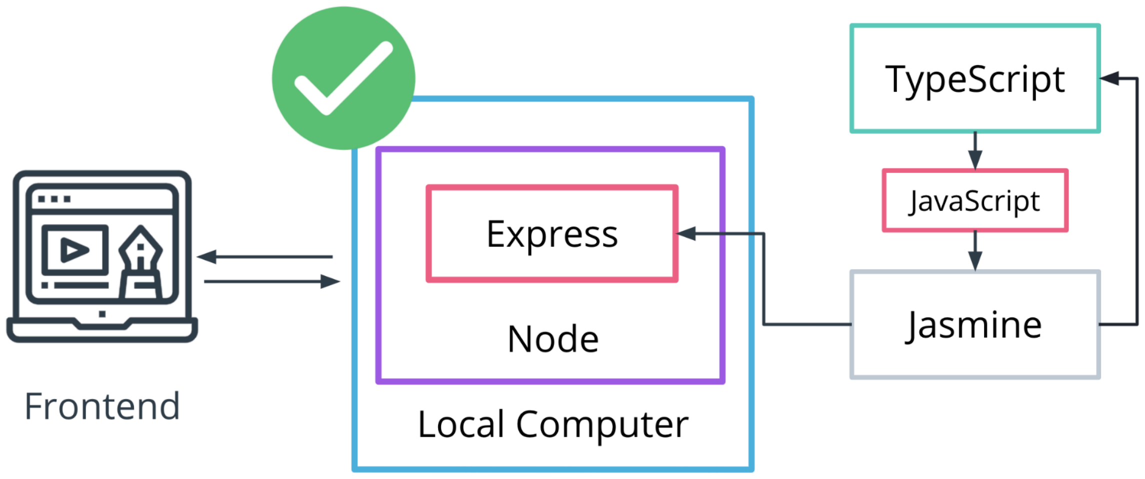
Foundations of Backend Development

1. Introduction to Backend Development with Node.js

    a. The backend has three parts
        i. **The server**: the computing resource that listens to requests from the frontend.
        ii. **The application**: code that runs on the server to process requests and return responses
        iii. **The database**: the part of the backend that is responsible for storing and organizing the data.

    b. The backend is responsible for processing the requests that come into the app and managing its data. That can mean different things for different apps. In a simple single-page application, the backend may only be needed to host the website. In other cases, the backend is also used to store, organize, and serve data. The backend also plays an important role in authentication, security, and scalability to ensure that the system has the capacity to handle all of the incoming requests.

    c. The image below shows the architecture of our application. The local computer will host the server and the right side shows that we will be using Jasmine, to ensure that we are writing performant code and catching errors and edge cases before making it to production.



2. Jasmine
    a. .Jasmine is a framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It runs in browsers and in Node.js. And it has a clean, obvious syntax so that you can easily write tests.

i. https://jasmine.github.io/

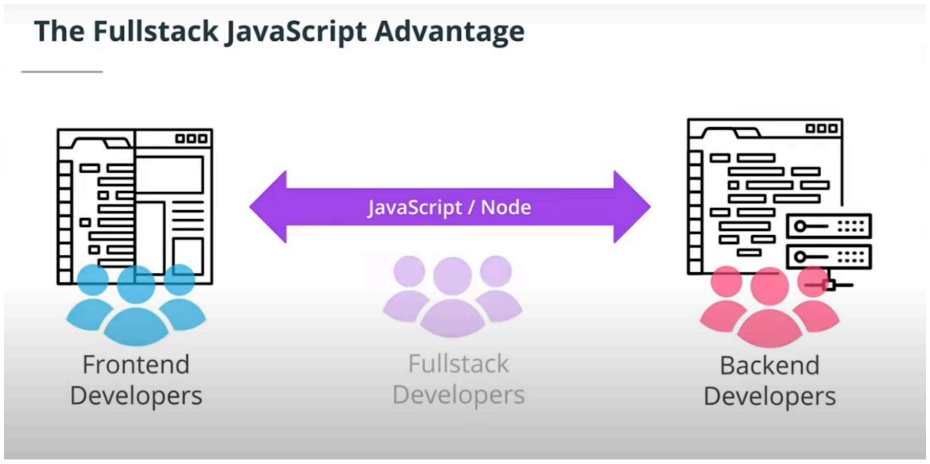
3. Containers
    a. Containers include the runtime, all configurations, and files needed to ensure that all individuals working on a project have the same environment regardless of the operating system or software installed globally. There are several choices to use to create a container for your application. Typically when working on an enterprise project, you will install a container software and run your project within that container which will also include a version of node.js that won't interfere with the version installed globally on your system.
    b. Docker containers
        i. https://www.docker.com/resources/what-container/

4. How to Structure a Development Team
    a. A development team's structure plays a significant role in a project's success. Surely, other factors like developers' expertise, experience, and talent are extremely important. Yet, proper management and teams' structure allow profiting from these factors as well as making the whole development process easier and faster.
        i. https://stormotion.io/blog/6-tips-on-how-to-structure-a-development-team/

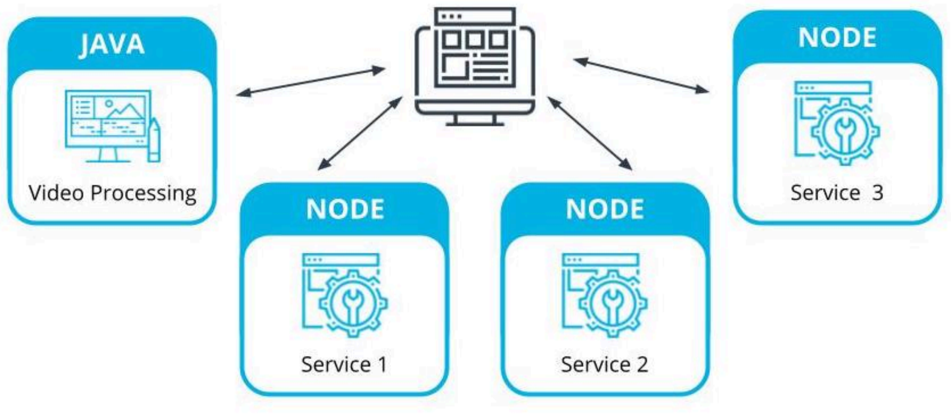
    b. Generalize Structure
        i. This approach implies building a development team of people with a highly diverse set of skills. Great results are reached thanks to the face-to-face communication and the cooperative effort of all members.
        ii. For instance, a front-end developer can also have some knowledge of back-end Java. Or a Project Manager can be familiar with UI design and help with this development part.
        iii.
    c. Specialist Structure
        i. This arrangement approach means that each team member is an expert in a certain programming language, framework, or technology, and thus, fully responsible for their part of development. You can create teams with their own hierarchy and structure to complete one part of the project. It all depends on the scope of work.
    d. Hybrid
        i. Hybrid project teams imply exactly what the name says. They have both people who focus on a product as a whole and can narrow their focus down when needed.

Getting Started with Node.JS

e. Node.js advantages
   i. Node.js allows for JavaScript to be used on the frontend and backend.
   ii. Node.js allows for easy application scaling and maintenance.
   iii. Node.js is easy to learn.



f. Node.js disadvantages
   i. Node.js is not well-suited for applications that require heavy processing and computation, like video processing, 3D games, and traffic mapping. In these cases, you can use a **microservice structure** to use a different language for the services that require heavy compute power and use Node.js for the rest.



Use Node.js in the Appropriate Modules

ii. **Microservices** are a piece of a larger application. In Microservices applications, the app is broken down into encapsulated parts that can be maintained individually.

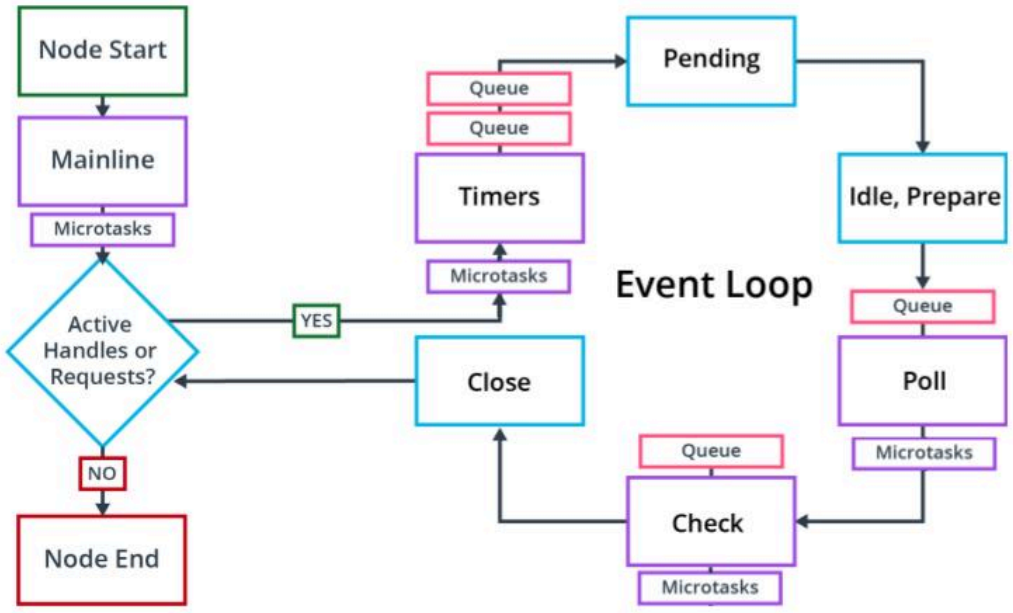
5. How to run JavaScript code
    a. Run JS code in REPL terminal
        i. Open powershell
        ii. Type node
        iii. Ctrl+d (to exit)
    b. Run JS code file
        i. node /path/to/file.js

6. Node.js globals
    a. Node.js and the V8 Engine
        i. The google chrome browser is built on top of an open source V8 engine. The V8 engine processes the following:
            1. **ECMAScript** (which is the JavaScript standard)
            2. **WebAssembly** (runs along JavaScript and allows us to run other languages, like C++/C# on a web platform)
        ii. The V8 engine translates the web assembly into byte code.
7. The Event Loop
    a. Nearly every Node.js feature is considered to be asynchronous (non-blocking). This means that we can request an API using promises and have our application continue running while that request is being waited for. But how does Node.js process that asynchronous request? Both the Browser and Node.js take advantage of something called the Event Loop. The Event Loop isn't an API or language; it's a process that runs anytime you have asynchronous code.

b. The Event Loop controls the order in which results (output) of asynchronous tasks (input) are displayed. Think of the Event Loop as the person working the door at an exclusive venue. That person lets people in based on a set of information provided by the venue. Your application is the venue, your asynchronous tasks are the people trying to get in, and it's your job to tell the door person how to do so. Once you become familiar with the Event Loop and the order in which Node.js handles tasks, you will control when those tasks occur in your application.
   i. Further reading:
      1. https://medium.com/the-node-js-collection/what-you-should-know-to-really-understand-the-node-js-event-loop-and-its-metrics-c4907b19da4c

c. Sixe phases of the event loop
   i. **Timers** - executes callbacks using timers. If there are timers set to 0 ms or setImmediate(), they will run here. Incomplete timers will run in later iterations of the loop.
   ii. **Pending** - *internal phase*
   iii. **Idle/Prepare** - *internal phase*
   iv. **Poll** - process I/O callbacks
   v. **Check** - execute any setImmediate() timers added in the Poll phase
   vi. **Close** - loop continues if there are more timers or I/O calls. If all timers and I/O calls are done, the loop closes and the process ends.

NOTE: process.nextTick(); will always run at the end of whichever phase is called and before the next phase.



8. Best Practices for Server Side Development
    a. In a professional environment, form becomes as critical to your application's success as the function. There are often many developers working on the same codebase. For this reason, there are some steps you can take at the start of a project to integrate quickly into a professional environment.

        i. Code Quality
            1. Use VS code extension **'Prettier'** for syntactic consistency.
            2. Use a linting tool like **'ESLint'** for making sure that your code follows style guides and that you aren't calling functions before they are declared.
        ii. Use ES6+ and Async/Await
            1. Our goal is always to make our code more maintainable and more readable. ES6 has become the standard and should be used.
            2. You may find yourself using promise chains if a module provides poor documentation on using Async/Await, or your team prefers it, but this is becoming less common.
        iii. Keep code small
            1. Applications should be scalable. Node.js is built for scalability. Keep services separate. Node.js encourages the use of modules. Take advantage. Don't make every function its own module, but it is reasonable to group similar functions as individual modules. If

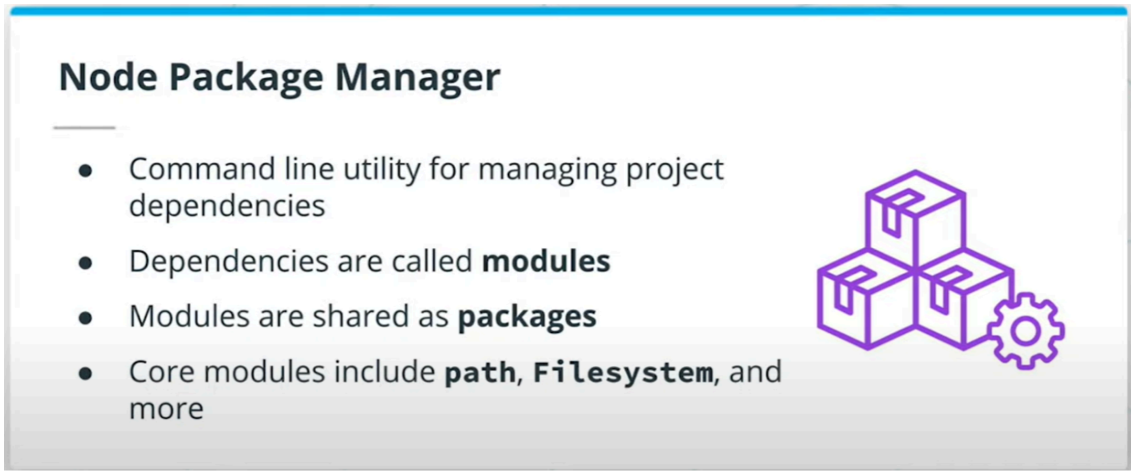
you create a module that can be used across your organization, NPM allows for creating **private npm packages** (discussed later).

2. With Node.js, there is no reason for your project to turn into an unmaintainable monolith. If your project is small and you never intend to grow it, perhaps monolith architecture works for you. Otherwise, it's worthwhile to learn more about **microservice architecture** and how it can improve an enterprise project.
    a. https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/

iv. Handle Errors
1. With respect to a server, the user should be presented with feedback about what has happened and a solution to continue using the application. The developer should be writing and presenting relevant error messages to locate edge cases, improve application reliability and debug.

v. Node.js best practices
1. https://github.com/goldbergyoni/nodebestpractices?tab=readme-ov-file

9. What is NPM ?
    a. **npm** is both a tool for managing project dependencies via command line and a website hosting more than 1M third-party packages that can be used for your project.



Node Package Manager

- Command line utility for managing project dependencies
- Dependencies are called **modules**
- Modules are shared as **packages**
- Core modules include **path**, **Filesystem**, and more

    b. Initializing npm and creating a package.json file
        i. Initializing npm will create a package.json within the root of your application folder containing general information about the project.
        ii. To initialize npm and go through all of the settings use:

```
1. npm init
```

        iii. To automatically select all default use:

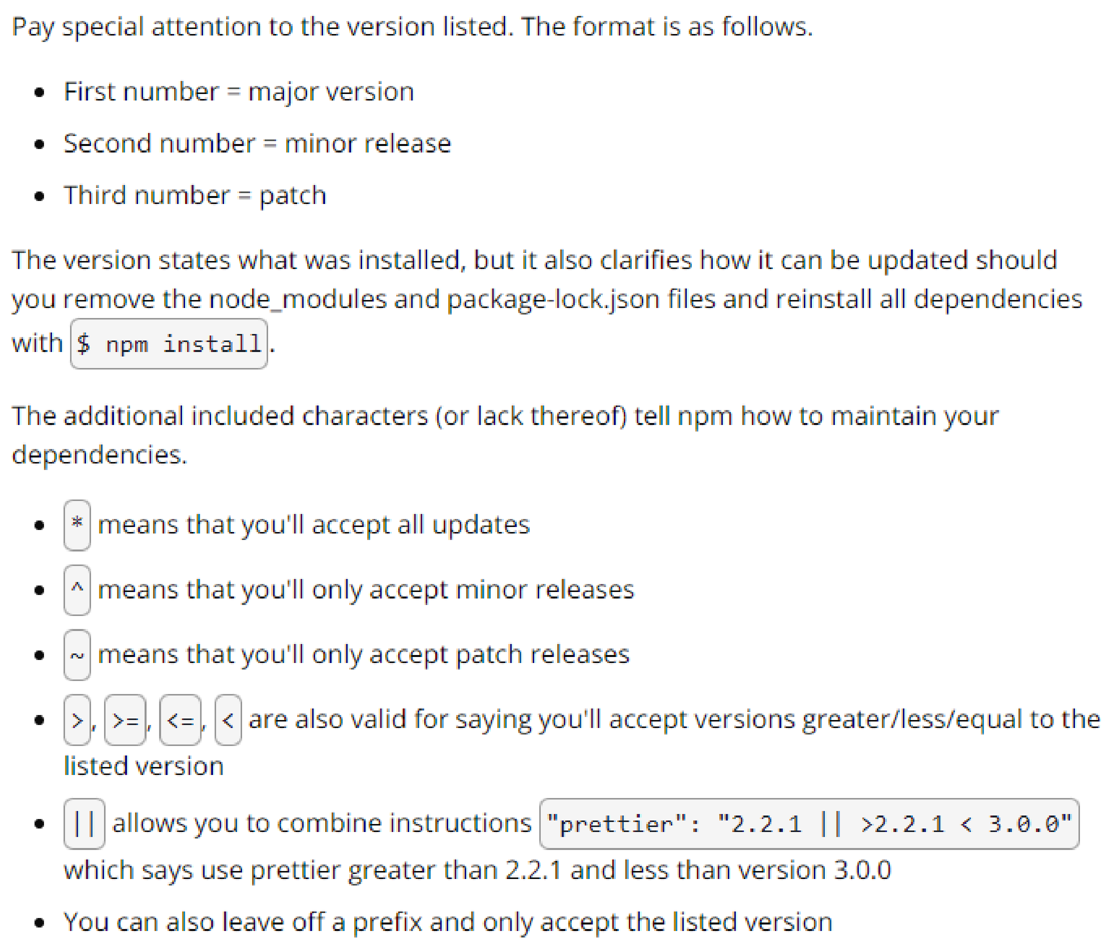
```
1. npm init -y
```

c. Applications will either include both dependencies and **devDependencies** or just **dependencies**. It is dependent on the team setting up the project. devDependencies are thought of as dependencies that are only necessary for development whereas dependencies are those dependencies used in both development and production.

d. An example would be needing TypeScript added as a dependency for development, but since it compiles to standard JavaScript to be used in production, TypeScript is not needed for production and therefore could be just a devDependency. Many teams find little use in separating but when learning, it can be a helpful practice to determine which dependencies are only being used in development vs which are also needed for production.

   i. `npm i module-name // install module to dependencies`
   ii. `npm i --save-dev module-name // install to dev dependencies`
   iii. `npm i --save-dev module-name@1.19 // install a specific version (1.19 here) of module`

   iv. Installing dependencies adds the dependency to your package.json file in the format:
      1.
      ```
      "devDependencies": {
         "prettier": "^2.2.1"
       }
      ```



Pay special attention to the version listed. The format is as follows.

- First number = major version
- Second number = minor release
- Third number = patch

The version states what was installed, but it also clarifies how it can be updated should you remove the node_modules and package-lock.json files and reinstall all dependencies with `$ npm install`.

The additional included characters (or lack thereof) tell npm how to maintain your dependencies.

- `*` means that you'll accept all updates
- `^` means that you'll only accept minor releases
- `~` means that you'll only accept patch releases
- `>`, `>=`, `<=`, `<` are also valid for saying you'll accept versions greater/less/equal to the listed version
- `||` allows you to combine instructions `"prettier": "2.2.1 || >2.2.1 < 3.0.0"` which says use prettier greater than 2.2.1 and less than version 3.0.0
- You can also leave off a prefix and only accept the listed version

e. Prettier is a code formatter that will ensure you're keeping your code consistent. It's commonly added to projects to ensure all members on a team are formatting in a consistent way such as always using semicolons, trailing commas, and single quotes. It can be configured to the preferred settings of the team and works well with additional tools like linting.

We are able to add it to a project with NPM by doing the following:

- Locate prettier on npmjs.com to get the install script and other information.
- Run the install script `npm i --save-dev prettier`.
- Add a prettier script to your `package.json` file. The script you choose can vary dramatically depending on the project. The one below will only overwrite files located in the src directory that are js files. You may need a **different script** depending on the project.

i. *`// example config file, path structure to check, and write fixes`*

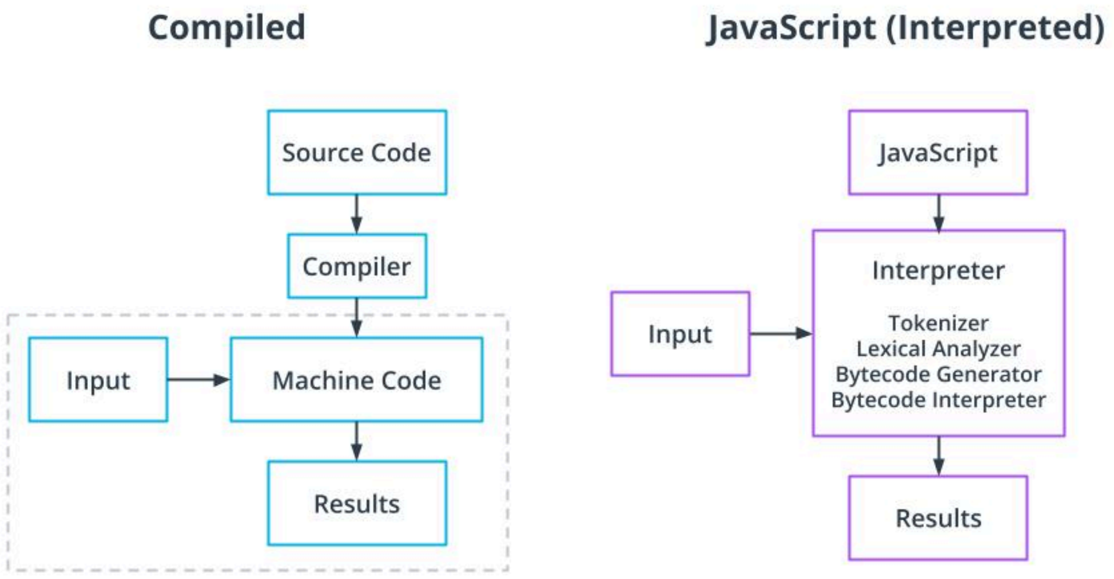
```
"prettier": "prettier --config .prettierrc 'src/**/*.js' --write"
                   // or
"prettier": "prettier --config .prettierrc src/**/*.js --write"
```

- Create a `.prettierrc` file for any custom configurations.
- Run `npm run prettier` to run prettier (or whatever you named your script).

10. Is Node.js single threaded
    a. JavaScript is single threaded, is Node.js?
        i. Compiled vs. Interpreted language

            1. Compiled language
                a. Language is written and compiled to machine code inside of an application
                b. Errors are detected during compiling
                c. The code won't compile until it's error-free
                d. Examples: C, Java, C++, Erlang, Go

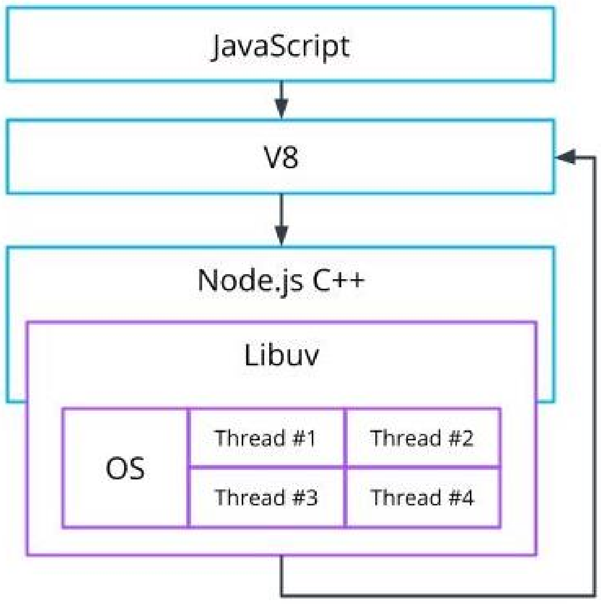
            2. JavaScript is an interpreted language
                a. Errors are found when the code is run
                b. The interpreter translates and runs code one statement at a time
                c. Interpreted code runs more slowly
                d. JavaScript runs the same on different systems because it is run in the browser not the CPU.



Compiled vs, Interpreted Languages

b. Node.js is an interpreter
   i. Node.js is mostly written in C++.
   ii. Many node modules also include some C++ code
   iii. Other modules include python or C

c. C Libuv library gives Node.js access to a thread pool
   i. Main thread runs async JavaScript
   ii. Libuv takes advantage of the operating system's asynchronous interfaces before engaging the thread pool
   iii. The thread pool is engaged for events that require more processing power including compression and encryption tasks
   iv. The default thread pool includes four threads



Libuv and Node.js

11. Why TypeScript is Important
    a. JavaScript is loosely typed and all variable's types are inferred. The type inference can lead to programmer errors, like trying to pass a string to a function that performs math operations.
        i. **sum(2,'2') => '22'**

    b. TypeScript is a compiled language, it compiles (transpiles) from TypeScript to JavaScript. In short, TypeScript is a static and string typed superset of JavaScript. *(JavaScript with types)*

    c. How to run .ts?
        i. **Option 1:**
            1. npm install -g typescript
            2. tsc .\<filename>.ts
            3. node .\<filename>.js

ii. **Option 2:**
   1. npm i typescript
   2. Create a folder for your new project
   3. npm init -y
   4. Update package.json

```
“scripts”:{
          “build”:”npx tsc”
}
```

   5. npm run build
   6. npx tsc - -init *(installs the tsconfig.json file)*

d. Update the default Typescript configuration file **tsconfig.json**
   i. You should always check your compiler options to note what you are transpiling to as well as your output directory. Common output directory names include ***dist***, ***build***, ***prod***, *and* ***server***

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "lib": ["ES2018", "DOM"],
    "outDir": "./build",
    "strict": true,
    "noImplicitAny": true,
  },
  "exclude": ["node_modules", "tests"]
}
```

   ii. You will see many more options available than what is above. Your application may require additional settings to be configured, but these are typically the main settings to start with. This file is where you can tell TypeScript how strict it should be while checking your code and what to ignore.

12. Importing modules
   a.
```
// Rename the module
import 'name' from 'module';

// Use destructuring to pull in specific functions when they
are exported individually
import {function, function} from 'module';
```

13. Exporting modules

   a.
```
// Export an individual function or other type of object in code
export const myFunction = () => {};
// Export a single item at the end
export default object;

// Export a list of objects
export default {object1, object2};
```

14. /…