node.js and express.js

 Summary of node.js and express.js, and their uses.

Recall

- What is Node.js?
- What is Express.js?
- What are the differences between promises and callbacks?
- What is the difference between blocking and non-blocking?

Overview

Node.js is a runtime that allows users to run Javascript on a server, rather than only a browser. It allowed web developers to write a full stack application in a single language.

Building a FS Web App

You visit a URL that points to a server.

When the request is recieved, you can use Node.js to handle that request and read a file (HTML) that is stored on the server's file system.

Then respond back to the client so they can view the file in the browser.

nvm (node version manager) is good to have on your system.

Using Node

One way to use Node.js is in `REPL` mode (Read Eval Print Loop)

- If you use the `node` command in the command line, it will allow you to write Javascript code, and will print the results.
- Use `Ctrl+C` twice to shut it down

The default entry point into a Node.js app is the index.js file (created manually). You can run this file with the following command: `node <file path>`

Node.js Runtime

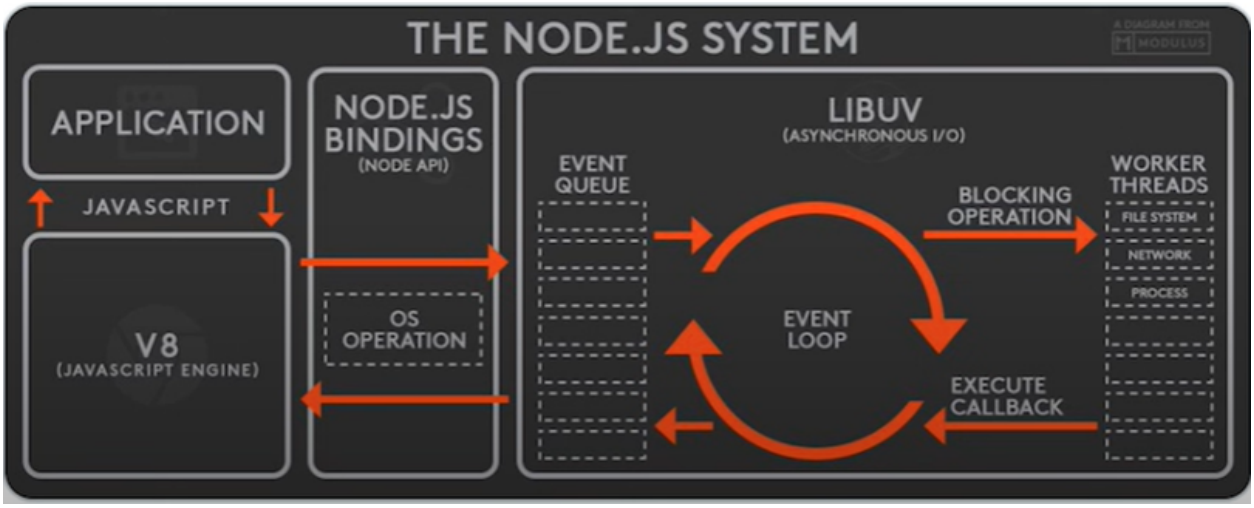
In most ways, Javascript runs in Node.js the same way it runs on the browser, but there are some important differences to know about.

Node has a handful of built-in identifiers (e.g console, global).

- `process` gives you access to the currently running Node.js process. You can use this to check the current platform/OS, grab an ENV variable from your server, etc.

Event

Node.js is often described as a asynchronous, event-driven, Javascript runtime. The runtime implements an 'event loop' just like a web browser, and allows Node.js to push intensive operations off to a separate thread, so that only fast, non-blocking operations happen on the main thread.



This makes Node.js very suitable for uses that require high throughput, like real-time applications, web servers, etc.

In most cases, you'll listen to events (which can come in many forms). You'll set up a callback function, a function that is called when an event occurs.

```
process.on('exit', foo(){
  // function code
})
```

You can also create your own event from scratch, and register a callback.

The event-driven style is helpful when you have a CPU intensive load.

File System

Blocking

Node.js has a built in file system module called `fs`. It can read, write, and delete files in its system, and can also execute its actions in either a blocking or non-blocking way.

💻 Anytime a function ends in sync, think blocking.

- sync === blocking

This means it will need to finish all of its work before any of the other code can run.

Non-blocking

Promises are another way to handle certain actions. They are cleaner to write, asynchronous, and non-blocking. Using `promises` gives a function that returns a promise when called. This function can be used with `async-await` syntax.

Modules

> 💻 A module is simply a Javascript file that exports its code.

The traditional way to import a module is to use the `require()` function.

Node has added a way to do the same with `import/export` syntax.

You can create your own modules to use, or use other modules.

Npm

> 💻 `npm` is a package manager for Node.js

`npm init -y` (or default options)

This creates a package.json file with meta-data about your project, and also the dependencies that are used.

The dependencies object allows you to manager multiple dependencies in a project, and reinstall them all at once on a different system.

The raw source code for the dependeny will be in the Node.js module's source directory. You should never need to modify this code.

Express.js

> 💻 This is a minimal web-application framework for node. It is one of the most popular third-party Node.js modules.

A framework that helps with the repetitive tasks of parsing URLs, setting up routes, etc.

`npm install express`

Once the package is installed, you an import it by name in the Javascript code.

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

An express app allows the creation of different URLs and endpoints that a user can navigate to in the browser. Code can be defined for the server to handle the different requests (GET).

Express provides two distinct parameters: request and response.

- REQUEST - The user's incoming data
- RESPONSE - Your outgoing data

Sometimes you will need to parse the request to authenticate the user or understand what they are trying to do.

To send a response in the form of an HTML file, you can use the `readFile()` function, creating a callback with the `response.send()` function to send the HTML.

- You can also send back an error message if the page is unavailable

```
app.get('/', (request, response) => {
  readFile('./home.html','utf8', (err, html) => {
    if(err) {
      response.status(500).send('This page is unavailable')
    }

  })

});
```

What's let is to tell the express app to listen to incoming requests. you do this by defining a port. This port will normally come from a Node.js ENV variable.

```
app.listen(process.env.PORT || 3000, () => console.log('App available on http://localhost:
3000'))
```

You can start this by running node in the current working directory. Once you open it in the browser you should see your HTML return back to you.

Callbacks can be difficult to work with as your app grows in complexity. It can often lead to `callback hell`, which is when you have stacks of nested callbacks.

A good way to avoid this is to use promises instead of callbacks.

```
app.get('/', async (request, response) => {
  response.send(await readFile('./home.html','utf8'));
});
```