FULL STACK WEB DEVELOPMENT PROGRAM

Lecture 6: Intro to Node.js & Express

BeaconFire

OUTLINE

- What is Node.js?
  Node Package Manager
  Module Systems
  Globals
  Core Modules (EventEmitter, File System, Stream, Timers)
  Event Loop

- Request & Response Cycle
  Express
  Postman
  Model-View-Controller

BeaconFire

- **Node.js**: an open-source, single-threaded, backend JS runtime environment that runs on the V8 engine.
  - Event-driven architecture (callbacks, event loop, EventEmitter) for asynchronous I/O (HTTP requests, file system)
  - https://nodejs.org/en/download/

- node -v
  node fileName.js

NODE.JS VS JS

	JavaScript	Node.js
What is it?	Programming language	JS Runtime Environment
Where is it used?	Frontend development, Node.js, General-purpose scripting	Backend (server) development
JS Engines	V8, JSCore, Spidermonkey, Chakra	V8
Web APIs (fetch, XHR, storage)	Can use in the browser	Not built-in

NPM & NPX

- What if we want to use third-party libraries/packages in Node.js?

- **Node package manager (NPM)**: a JS package manager that provides a command-line interface (CLI) for accessing and installing packages from the **npm registry** (a list of public/private packages created by other developers).
  - Included when you install Node.js
  - Online registry: https://www.npmjs.com/

- npm -v
  npm init -y

- **Node package eXecute (NPX)**: a JS package runner that can be accessed as a CLI command (don't need to install a local copy).
  - Used when you only need to use said package once
  - npx create-react-app

PACKAGE.JSON

- **Package**: a folder tree that contains a describing file (package.json) and all contents, excluding nested packages.

- **“version”**: tracks the version of this package (mainly for uploading on npm)
  - [major].[minor].[revision]

  “main”: specifies a single entry point of the application (the default module when loading this package)
  - Supported in all versions

  “exports”: specify multiple entry points (modules that can be loaded)
  - For versions after Node10

  “scripts”: shortcuts for executing long terminal commands
  - npm run scriptName

JS MODULE SYSTEM

- **Module system**: a standard for implementing modules in JS runtime environments.
- CommonJS
  - Backend: Node.js
- Asynchronous Module Definition (AMD)
  - Frontend: RequireJS
- Universal Module Definition (UMD), a combination of CommonJS + AMD
- ES Modules

COMMONJS

- **CommonJS**: the standard module system built into Node.js.
  - Not supported in browsers

- **require()**: a function used to import modules synchronously (built-in, user-defined, third-party).
  - May be called anywhere in the code

- **module.exports**: an object containing the code that will be exported.

ES MODULES VS COMMONJS

	ES Modules	CommonJS
Loading modules	asynchronous Pre-parses file to load modules first	synchronous Loads modules on demand
Import & export	import export	require() module.exports
Import usage	At the beginning of the file	Anywhere

```
// CommonJS
// script.js
console.log("CommonJS: script.js");
const PI = require('./utils');
console.log(PI);

// ------------------------------------
// utils.js
console.log("CommonJS: utils.js");
module.exports = 3.14;
```

```
// ES Modules
// script.mjs
console.log("ESModules: script.mjs");
console.log(PI);
import { PI } from './utils.mjs';

// ------------------------------------
// utils.mjs
console.log("ESModules: utils.mjs");
export const PI = 3.14;
```

```
> lec06-node-express-demo@1.0.0 cjs
> node 1-modules/script

CommonJS: script.js
CommonJS: utils.js
3.14

> lec06-node-express-demo@1.0.0 esm
> node 1-modules/script.mjs

ESModules: utils.mjs
ESModules: script.mjs
3.14
```

https://www.sitepoint.com/understanding-es6-modules/
https://blog.logrocket.com/commonjs-vs-es-modules-node-js/

CONFIGURING ESM VS CJS

- **“type”**: determines whether to load .js modules as CommonJS or ES Modules.
  - default: commonjs

- CommonJS (default)
  - Option 1: Change the file extension to **“.cjs”**
  - Option 2: (in nearest package.json) **“type”=“commonjs”**

- ES Modules
  - Option 1: Change the file extension to **“.mjs”**
  - Option 2: (in nearest package.json) **“type”=“module”**

PACKAGE.JSON (2)

- **”dependencies"**: packages that must be installed to run the application.
  - **npm i**: checks package.json and locally installs all dependencies.
  - **npm i packageName**: locally installs a package, automatically adding to package.json
  - **npm i -g packageName**: globally installs a package where node is located, doesn’t need to be a dependency

- **“devDependencies”**: packages that are only used in development and are not necessary to run the application.
  - **npm i -D packageName**: locally installs a package and adds it to devDependencies

- **node_modules**: the folder containing all locally-installed packages, enabling you to import them.
  - Always regenerated when using “npm install”
  - Never include them in zipped files or when uploading to code-hosting services

NODEMON

- **nodemon**: a package that provides a CLI command that starts a node application and automatically restarts after detecting any changes in the directory.
  - npm i -D nodemon
  - nodemon index.js

GLOBAL SCOPE

- Browser: top-level scope is global scope (window).
  Node.js: top-level scope is scoped to that module (file).

- **global**: the global namespace object.
  - Built-in global objects: **console**, **process**

- **Module-scoped objects**: objects that appear to be global but are locally-scoped to the module.
  - **module**: the object representing the current module
  - **exports**: a reference to "module.exports" (shouldn't be reassigned)
  - **require()**, **__dirname**, **__filename**

- https://nodejs.org/api/modules.html#the-module-object
  https://www.geeksforgeeks.org/what-are-the-global-objects-of-node-js/

PROCESS

- **Process**: the execution of a program.
  - Has its own private memory, executable code, threads, …

- **process**: a global object that gives information and control over the current Node.js process.
  - .env, .exit()

NODE.JS API

- **Node.js Core Modules**: built-in libraries that you can access through imports (require).
  - package.json not needed

- Events, fs, path, stream, timers

- Errors: Inherit from standard JS <Error> class.
  - https://nodejs.org/api/errors.html#errors
  - https://nodejs.org/api/errors.html#common-system-errors

EVENT EMITTER

- **EventEmitter**: a module that allows objects to communicate with each other through named events and listeners.
  - Works very similarly to browser event listeners
  - Event handlers are executed synchronously
  - Node.js core modules are built on this (HTTP handlers, responses, streams)
  - https://nodejs.org/api/events.html#events

- **Publisher-subscriber (pub-sub)**: a software architecture design pattern where publishers emit events to their subscribers, who react by doing some actions.

- **emitter.on('eventName', eventHandler)**: defines a handler that is executed whenever the event is triggered
  emitter.once('eventName', eventHandler): defines a handler that is executed only on the first event trigger
  emitter.emit('eventName', arg1, …, argN): triggers/fires the event

```
const EventEmitter = require('events');
const eventEmitter = new EventEmitter();

// Emitting events with and without registered handlers
eventEmitter.on('logMessage', () => { console.log('logMessage event: A custom event was fired.'); });
eventEmitter.emit('logMessage');
eventEmitter.emit('logMessage');
eventEmitter.emit('unregisteredEvent');
```

FILE SYSTEM

- **File System**: a structured representation of data that manages how and where your files are stored.

- **fs**: a module for interacting with your file system (read, write, update…).
  - 3 API types: synchronous, asynchronous callback, promises

  path: a module that provides ways to work file and directory paths.

- **fs/promises**: the promise-based API for asynchronously interacting with the file system.
  - **stat(path)**: returns details about the file at this path, throwing an error if it doesn't exist
  - **readFile(path)**: read all the contents of the file at this path at once
  - **writeFile(path, data)**: write data to the file at this path, replacing it if it already exists
  - **appendFile(path, data)**: add data to the file at this path, creating one it if it doesn't exist
  - **rm(path)**: removes the file at this path

Method	Description
basename()	Returns the last part of a path
delimiter	Returns the delimiter specified for the platform
dirname()	Returns the directories of a path
extname()	Returns the file extension of a path
format()	Formats a path object into a path string
isAbsolute()	Returns true if a path is an absolute path, otherwise fals
join()	Joins the specified paths into one

```
const filePath = path.join(__dirname, '/test.json');
```

STREAM

- **Stream**: an interface for managing large amounts of data more efficiently by processing them in smaller chunks at a time.
  - Easier than trying to process all data at once (need to store in memory)
  - https://nodejs.org/api/stream.html#stream

- **Writable**: streams that you can write data to
  - HTTP client request & server response, fs write stream

  Readable: streams that you can read data from
  - HTTP client response & server request, fs read stream

  Duplex: streams that can be read from and written to
  - net.Socket

  Transform: Duplex streams that can modify the data during read and write operations

- "fs/promises" **open(path)**: creates a FileHandle object (extends EventEmitter) based on the file at the specified path.
  - **createReadStream()**: a FileHandle method that creates a readable stream
  - **createWriteStream()**: a FileHandle method that creates a writable stream

TIMERS

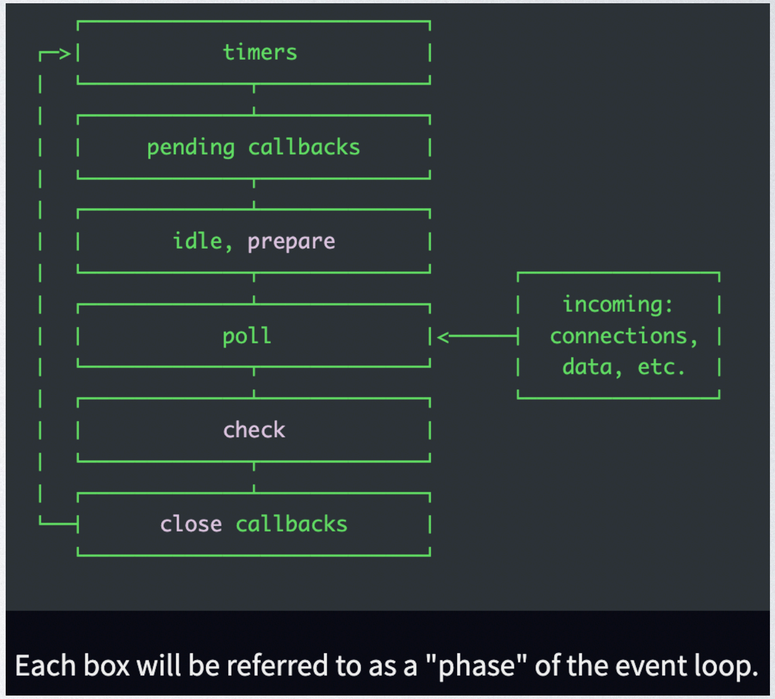
- **timers**: a module that exposes global scheduling functions, implemented differently from the Web APIs.

- **Immediate**: an object created from the immediate scheduling functions.
  - setImmediate(callback): schedule a function to be executed right after the I/O event callbacks in the event queue
  - clearImmediate(immediate)

- **Timeout**: an object created from the timeout scheduling functions.
  - setInterval(callback, delay), clearInterval(timeout)
  - setTimeout(callback, delay), clearTimeout(timeout)

EVENT LOOP (NODE)

- Macrotask Queues
  - **Timers**: setTimeout/setInterval callbacks
  - **Pending callbacks**: I/O callbacks
  - **Idle, prepare handlers**: only used internally
  - **Poll**: check for I/O events & execute their callbacks (HTTP requests, file system)
  - **Check handlers**: setImmediate callbacks
  - **Close callbacks**: close events (socket.on('close',…), process.exit())
- Microtask Queues
  - **process.nextTick(callback)**: schedule a function to be executed right after the current phase completes
  - promises…
- https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/



NODE VERSION MANAGEMENT

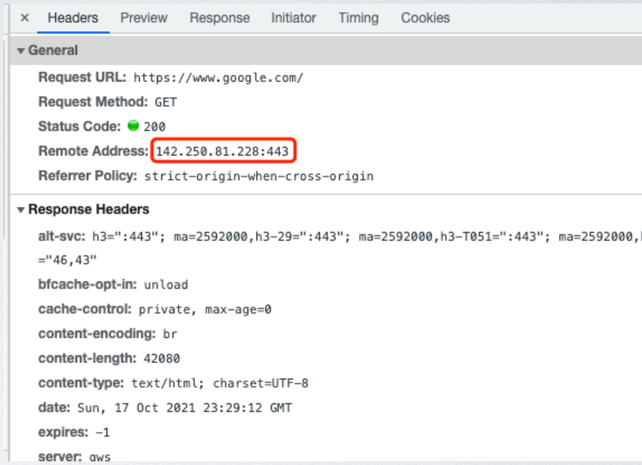
- **n**: an npm library that must be installed globally and will copy the specified node version to /usr/local/bin.
  - Does not affect global modules (can have issues when switching)
  - https://www.npmjs.com/package/n

- **Node Version Manager (NVM)**: a standalone package that adds different versions to a specified PATH.
  - Does not support Windows OS, unless you configure WSL or install other programs
  - https://github.com/nvm-sh/nvm

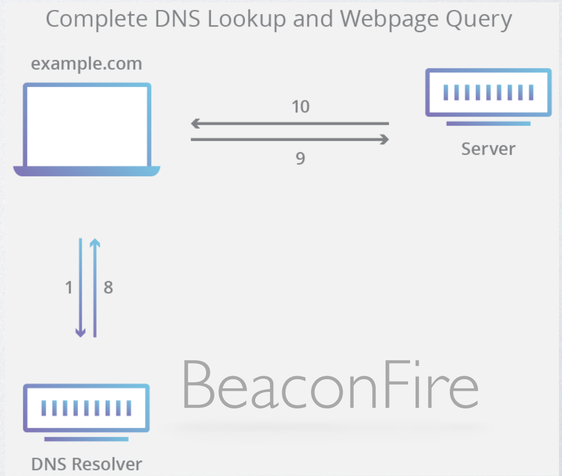
REQUEST & RESPONSE CYCLE

- What happens after I type google.com in the browser?

- **Domain Name**: a unique name that identifies a specific website.
  Internet Protocol (IP) Address: a unique string that consists of four numbers (0-256) separated by periods.
  - Assigned to all web servers and computers connected to the internet
  Domain Name System (DNS): a mapping of domain names to server IP addresses.



- 1. Browser sends a GET request to the DNS (type in URL, JS sends AJAX request)
  - DNS resolver processes the request and returns the server IP address
  - Browser sends HTTP request to that IP address
  2. Server receives the request, processes it (reads computer IP address), and sends back a response containing the html file
  3. Browser gets the response, parses the HTML, and displays the webpage



- https://www.cloudflare.com/learning/dns/what-is-dns/

SERIALIZATION & DESERIALIZATION

- How is my request body compressed and sent over the network to my server?
- **Serialization**: the process of converting an Object into a stream of bytes.
  - Data becomes platform-independent, easier to transfer over the network or save it
- **Deserialization**: the process of converting a stream of bytes into the original Object.

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40</age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
     {
          "employees" :  [
          {
               "name" : "James Kirk",
               "age" : 40,
          },
          {
               "name" : "Jean-Luc Picard",
               "age" : 45,
          },
          {
               "name" : "Wesley Crusher",
               "age" : 27,
          }
                              ]
     }
}
```

NODE.JS SERVER

- **Strengths**: highly scalable (handles many requests concurrently without blocking), efficient (V8), community support, same language as frontend, real-time applications
  https://stackoverflow.com/questions/34855352/how-in-general-does-node-js-handle-10-000-concurrent-requests
  Weaknesses: CPU-intensive computations, deprecated APIs, asynchronous programming, poor quality npm libraries

- **Express**: lightweight, unopinionated, most popular JS web framework
  - Built-in tools for handling HTTP requests, routing, template engine integration, port configuration, middleware
  - npm i express
  https://expressjs.com/en/4x/api.html#express
  https://heynode.com/tutorial/what-express-nodejs-framework

EXPRESS SET-UP

- **Middleware**: functions with access to the incoming request and outgoing response objects.
  - Usually transforms them before getting to their endpoint
  - Executes in the order they're applied

- **Port**: communication endpoints that separate web traffic for different services, ranging from 0 to 65,535.
  - **0-1024**: reserved (**80**:HTTP, **443**: HTTPS)

- **app.use(path, mw)**: applies middleware on a specific path.
  app.METHOD(path, cb): creates a route that executes the callback every time the application receives an HTTP request with a matching method and path.
  app.listen(port, cb): binds this host & port and waits for incoming requests.

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

// Applying middleware
app.use('/', express.json()); // parse requests with JSON payload/body

// Route handling for HTTP requests
app.get('/', (req, res) => {
  res.send("hello");
});

// Make server start listening for requests
const port = 3000;
app.listen(port, () => {
  console.log(`Server is up on port ${port}: http://localhost:${port}`);
});
```

STATIC FILES

- **Static files**: files that don't change

  - Browsers automatically download them when displaying a webpage (HTML, CSS, JS, pictures, icons)

  - By default, express servers do not let browsers access any files

  https://expressjs.com/en/starter/static-files.html

  https://www.webmound.com/serve-static-files-in-express-nodejs/

- **express.static(path)**: middleware that defines a folder at the path as publicly accessible

EXPRESS ROUTING

- **Routing**: describes what code to execute based on each request's URL and HTTP method.
  - /: root path
  - *: matches all paths

- **express.Router()**: a function that creates modular route handlers.
  - Created in separate .js files and exported
  - Imported by the main app and mounted on a path as middleware (app.use)

- https://expressjs.com/en/guide/routing.html

```
const router = require("express").Router();

router.get('/route1', (req, res) => {
  res.status(200).send(`Hello, this is ${req.url}.`);
});

// Query strings/parameters
router.get('/route2', (req, res) => {
  console.log("GET /route2", req.url, req.query);
  res.status(200).send(`Hello, this is ${req.url}.`);
});

// Route parameters
router.get('/route3/:userId', (req, res) => {
  console.log("GET /route3/:userId", req.url, req.params);
  res.status(200).send(`Hello, this is ${req.url}.`);
});

module.exports = router;
```

```
// Importing other routes
const otherRoutes = require('./otherRoutes');
app.use('/other', otherRoutes);
```

EXPRESS REQUEST AND RESPONSE

- **Query strings/parameters**: data sent in the GET request URL after the ? (ex: **/?page=2&limit=3**)
  - Usually for filtering data, not included in the route path

  Route parameters: named segments of the URL that represents values at that position. (ex: **/profile/:userID**)
  - userID would be mapped to some value in the URL (ex: **/profile/1234**, userID=1234)

  https://stackabuse.com/get-query-strings-and-parameters-in-express-js/

- **Request (req)**: the HTTP request that the express app receives.
  - **req.body**: key-value pairs of data in the request body (only if the parsing middleware was applied)
  - **req.params**: object that maps the route parameters to their values
  - **req.query**: object containing the query string parameters in the route

- **Response (res)**: the HTTP response that the express app sends back.
  - **res.status(status)**: set the response HTTP status code
    - status: number or string
  - **res.send(data)**: send out the HTTP response
    - data: string, object, boolean, array,…
  - **res.json(data)**: convert the data to JSON string, set content-type to JSON, and send the response

  https://stackoverflow.com/questions/44692048/what-is-the-difference-between-res-send-and-res-write-in-express

```
const router = require("express").Router();

router.get('/route1', (req, res) => {
  res.status(200).send(`Hello, this is ${req.url}.`);
});

// Query strings/parameters
router.get('/route2', (req, res) => {
  console.log("GET /route2", req.url, req.query);
  res.status(200).send(`Hello, this is ${req.url}.`);
});

// Route parameters
router.get('/route3/:userId', (req, res) => {
  console.log("GET /route3/:userId", req.url, req.params);
  res.status(200).send(`Hello, this is ${req.url}.`);
});

module.exports = router;
```

RESPONSE STATUS CODES

- https://docs.aws.amazon.com/AmazonSimpleDB/latest/DeveloperGuide/APIError.html
- https://kinsta.com/blog/http-status-codes/

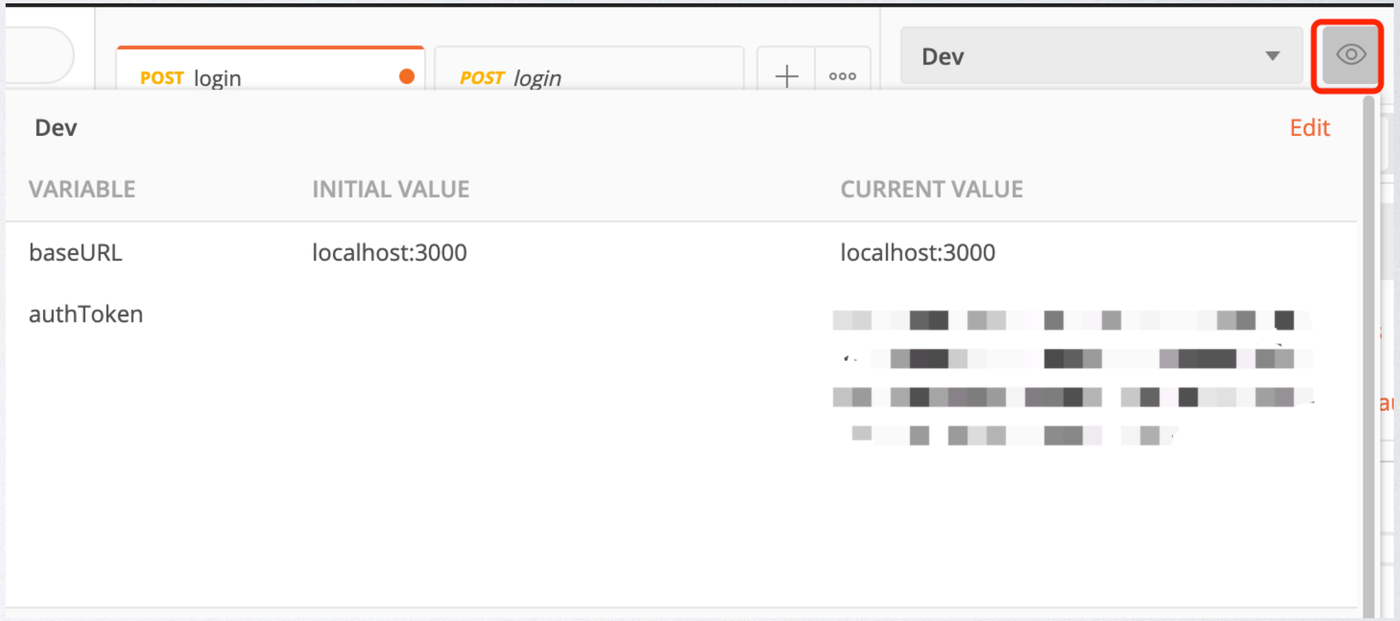
GET VS POST

	GET	POST
Sending data	Added to URL in name-value pairs	Added to request body
Restrictions	URL length is limited	Request body data size isn't limited
Security	Visible in browser URL	Safe
Use-cases	Retrieving resources Navigating to new pages	Updating database Retrieving resources with specific criteria in body

POSTMAN

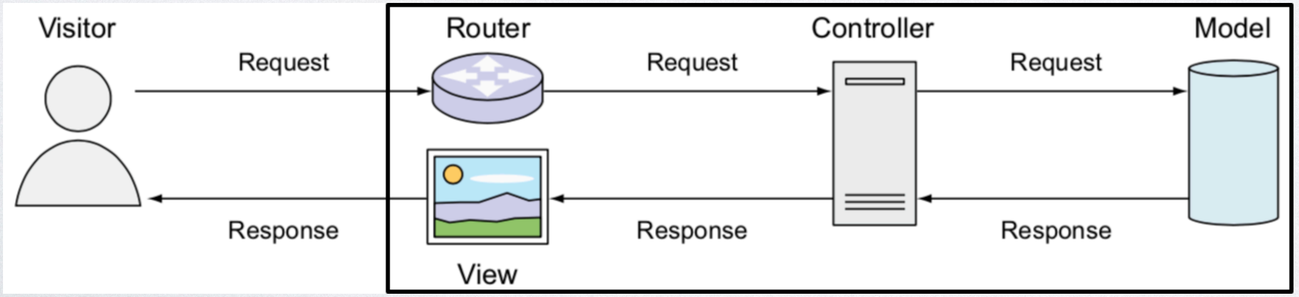
- **Postman**: an API platform that lets you configure and send HTTP requests to your server during development.
  - Use variables to reuse values across requests and scripts

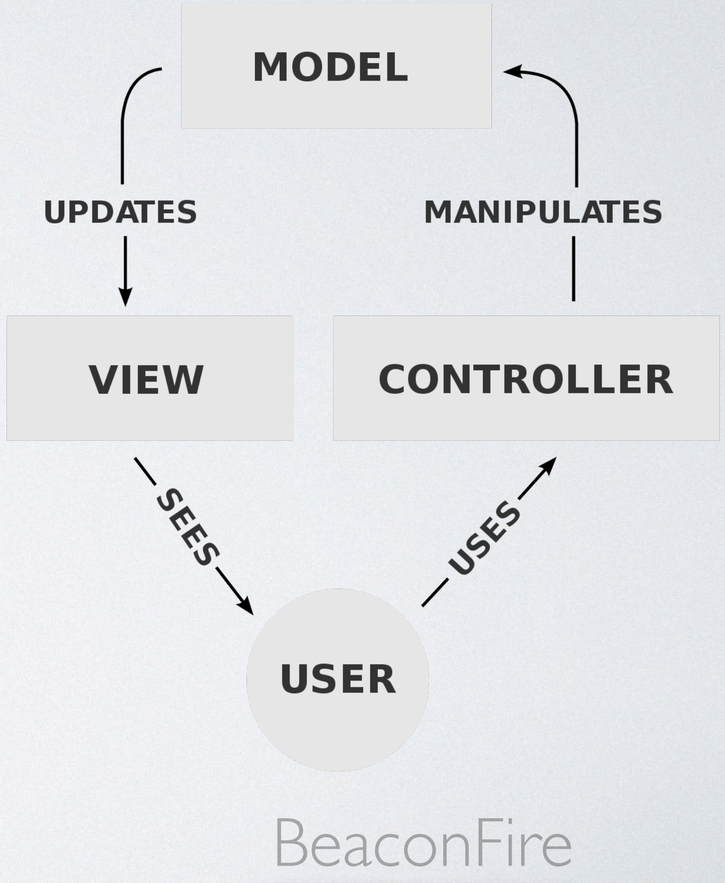
- https://www.postman.com/



MVC

- **Model View Controller (MVC)**: a design pattern & application architecture where you divide it into three components:
  - model: logic related to the database
  - view: logic related to the UI
  - controller: processes incoming requests, interacts with data models, and updates the view





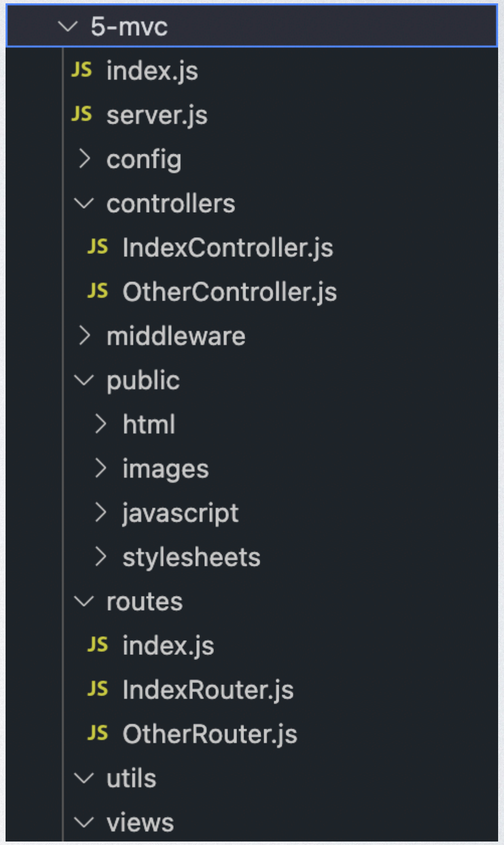
BeaconFire

MVC FILE STRUCTURE

- **index.js**: the entry point of the server/application, usually executed with "node"
  server.js/app.js: a module that sets up the server application (imported into index.js)
  - Often when the server will be run by another script (testing)

- **models/**: abstract representations of data & their relationships
  views/: the files used to display the webpage/UI
  routes/: maps the request to the controller
  controllers/: handle the request & response logic

- **config/**: settings for connecting to the database, environment variables, API keys, credentials…
  middleware/: functions that intercept requests (auth, logging…)
  public/: static assets available for download by the browser
  utils/: code/shared logic that is reused throughout the application
  test/: any testing files



ANY QUESTIONS?