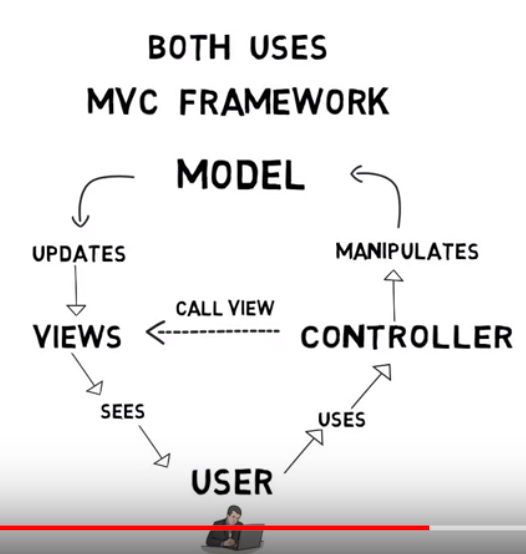
Node.js Tutorial (Back End)



\*Other Stuff\*

* ASP.NET, Rails – Synchronous out of the box (by default)
* Single thread that can only serve 1 client at a time
* Server may run out of threads when handling many concurrent clients at the same time
* Must either add more hardware (another server) or simply have the clients wait for a free thread

**Back End**:

* Provides backend services – API’s - Application Programming Interface
* Event driven, Non-blocking I/O model
* Data intensive real-time applications

Architecture:

* Node is NOT a programming language, or a framework.
* NODE IS A RUN TIME ENVIRONMENT FOR EXECUTING JAVASCRIPT CODE
* Every browser contains a JS engine to run JS code
* The browser provides a runtime environment for javascript
* Until 2009 the only way to run JS code was inside of a browser
* Ryan Daul – embedded googles V8 engine inside of a C++ program 🡪 created Node.exe
* Node.exe –
* Contains JS engine to run the JS code
* Contains objects (Modules) that provide an environment for our JS code

(different than the objects in browsers like ‘document’, ‘window’)

First Application:

* Check if node is installed 🡪 node --v
* Write JS code in text editor – name file app.js
* Go to directory containing the file and type command ‘node app.js’ into cmd prompt
* Node takes this JS file and gives it to the v8 engine for execution
* Output is displayed on the cmd prompt console

**Node Module System** –

**Global Object** – like the window object in browsers

* Variables & Functions declared are not automatically added to the scope of the global object
* only available inside that JS file in which they were declared
* This behavior is unlike JavaScript where they are in fact added to the global scope of window
* Problem with this – usually JS is broken up into multiple files, so:
* Defining a method or var with same names across multiple files – overrides
* \*Avoiding this in JS is easy by creating a var Home = {}; & scoping everything in that file off of that Home object\* (personal note)

**Node’s Modular System** –

* Prevents variables & functions from being added to the global scope and overriding each other
* Variables & functions with the same name used in different JS files but in the same app would override each other’s definitions
* Module: every JS file inside of a Node application is considered a module
* The variables & functions declared in that file are scoped to that file only
* O-O terms 🡪 they are private, not available outside of that file
* Main Module: every Node app has at least 1 module called the main module

**Export & Load Modules** –

* To make methods & variables accessible in other JS files)
* must explicitly export it & make it public
* Export desired fields in the file they are declared
* **module.exports.log = log**;
* this creates a new log object in the exports scope & sets it to the log function/var defined in the JS file
* Load these exported fields in the file in which you wish to access them in
* **const logger = require(‘name of JS file that exported’)**;
* this returns the object that was exported from the target module
* **logger.log**
* now has access to the log function
* \*Best to store it in a const rather than a var to catch errors at compile time rather than at runt time\* try JS Hint to find JS errors

Module Wrapper Function – 33:00

\*Node does not execute our code directly\*

* Node wraps the code of our module inside of a function

Built in Core Modules:

* os
* fs
* events
* http

**Node.JS** Runtime – node.exe

* A JavaScript runtime built on Chrome’s V8 JavaScript engine
* Node is essentially JS running on the server
* The runtime environment provides the built-in libraries that are available to the program at runtime (during execution)
* Node.js uses an Event Driven, Nonblocking I/O model
* Asynchronous (non-blocking) – highly scalable
* Single thread – handles multiple requests simultaneously
* Serves client (request) – might need to query a database
* Can immediately serve another client while the first query is being executed & returned
* Database prepares the response and put it in an Event Queue
* Node constantly monitors this queue in the background to process events
* \*This makes Node ideal for I/O intensive apps – a lot of disk/network access
* \*Not ideal for CPU-intensive applications

Traditional server-side technology – ex: PHP application on Apache server

* Synchronous
* Each connection / request spawns a new thread which takes up system memory
* Eventually will max out
* Essentially needs to wait for one process to stop before starting the next

REPL –

* Read, Eval, Print, Loop
* Allows us to run JS in the Command Line
* JavaScript without the browser / DOM functionality

NPM package manager –

* Allows us to install Packages & Modules for node
* Execute ‘npm init’ in cmd prompt

Package.JSON File –

* Main manifest file you must have in your node apps
* Goes to root of your package/application
* Tells the NPM how your package is structured & how to install it

Node.js most useful for:

* Restful API’s, Backend Applications
* Anything NOT CPU intensive

Building a Basic Web Server –