**Week 2 - Preview**

**Overview**

Week two will switch gears and focus on Back-end Development. We will discuss the client-server relationship and how the server responds to client's requests. We'll talk about some of the complexities of writing code server side and it's many benefits. We'll quickly see the need for databases as well and some of the advantages they provide us in developing applications.

**New Technologies + ELI5: Description**

\* Node - Run JavaScript on your Computer \* Express - Make web servers easily in Node! \* PostgreSQL - Store data in your hard drive in an organized way. \* Sequelize - Writing SQL is hard, make it easier with JavaScript functions.

**Week 2 Problems and Associated Solutions.**

Some problems we will discuss and accomplish:

1. *How can we run JavaScript on our machines outside of a web-browser and why do we care?*

Currently we are able to execute JavaScript because our browsers have support for it via the V8 engine. Unlike Java, or C++, or many other programming languages which must go through a step known as *compilation* to run (recall all computers at the end of the day only understand binary electrical impulses), JavaScript requires an engine that instead of pre-compiling will compile as it is being used. This lead people for many years to think JavaScript will never be more than a rinky-dink browser language for making some cool animations (hence "Script" even in it's name - Scripting languages are thought of as small-task languages not application languages). Anyway, putting JavaScript on our computer would certainly be a big deal as it would allow JavaScript to execute directly on your machine! Well why do we care? Applications on your machine have access to all of your computers resources, your file system, network, etc. allowing us the perfect environment to develop web servers. So you can thank Ryan Dahl, the creator of node, who took that V8 engine, performed some voodoo magic, and *voila* you can now execute JavaScript on your computer with the help of node.

1. *How can I use this Node thing to build a server?*

Ryan Dahl thought carefully about this when designing node and implemented a built in module known as HTTP. To be a good server, you just need to be able to make yourself available over the internet, listen for requests, and generate responses. This is the major application networking protocol used over the internet and it's supported out-of-the-box in node. We'll quickly find that although this solution is quite powerful and certainly *fast*, other developers created a now industry-standard library known as *express* which takes care of much of the verbosity around node's built-in HTTP module which we will be using throughout our time developing here at Fullstack.

1. *I've heard Facebook is running like 65 million lines of code! How will I be able to stay organized when I start writing a lot of code?*

We'll quickly see our number of code files per project go up very quickly. Node will provide a system of allowing us to create *modules* in which we can separate code into separate files but allow them to interact as if they were not! Writing modular code is an important step in being a mindful developer.

Advanced Note (feel free to skip): We're also going to talk about how the front-end code (the code we've been writing for the browser) and Back-end code (our server code) are essentially going to be two separate pieces of our application with no direct access to each other. This seems to become unclear because we are using JavaScript in both places. If you consider the fact that our Back-end code could've been written in Ruby on Rails, it becomes clear that the browser-side client code is essentially *unrelated* to our back-end code. Because the two are written in the same language, we'll need to make this distinction ourselves! It is important to remember the statement that *your server is in Norway and your Client is in New York* when asking yourself can these two files share code?

1. *If I'm building a twitter clone and I save users tweets as an array of objects, what will happen if I restart my Server?*

We're going to quickly realize that we are constantly messing up our code! We've often used some sort of *transient* storage as a means of saving data but as we even see with our game-of-life, a simple refresh takes us back to square 1. This becomes a problem as could you imagine if every time twitter made an update to it's code (forcing it to restart the server aka refresh) it lost ALL OF THE TWEETS! To solve this, we'll need some sort of *persistence* and this is how we will be introducing Databases

1. *What kind of Database will we be using and how will I talk to it?*

As we move through next week we'll begin communicating with our persist data store by using SQL a common query language to communicate to relational databases (we'll explain that term next week ;-)). The type of data-store we chose here to use is PostgreSQL for industry-wide adaptation, open-source support, and support of some complex data types. Toward the end of next week, we'll find SQL often verbose and we will use a special JavaScript library called Sequelize which will convert JavaScript into SQL for us (amongst other things) allowing us to build truly *Fullstack* Javascript applications! (AHHHH THEY SAID FULLSTACK LOL ROFL LMAO SOOOO FUNNY)

1. *Is there anything I should prioritize as prep for week 2?*

YES. Listen to the first episode of Fullstack's Podcast Tech'd Out, "1. The Internet, Browsers, and How JavaScript Became Trendy". IF you're feeling really bold check out the episode on Servers as well.

Spotify: <https://open.spotify.com/episode/3Iq2pJNcbm2I0xmpRBtvoN> Apple Users: <https://itunes.apple.com/us/podcast/techd-out/id1375045443> Everyone Else - 10 other platforms: <https://anchor.fm/techdout/episodes/1--The-Internet--Browsers--and-How-JavaScript-Became-Trendy-e1bubb>