Introduction to

Web Development

by John Wolf

1

Web Development

“Code it and let it live”
– John Wolf

Let’s get serious and maybe get with the 21st Century. Screens are everything, paper is just pulp.

World Wide Web - Protocols

Knowledge is king. It doesn't matter how you get it, from a cereal box, an MIT online course, slogging through a four year university gig, or on-the-job training from a web guru, just get it. YouTube gurus have well established themselves as credible resources. Internet based academies and bootcamp courses are all good, not to mention all the major universities, but what I want to do here is lay a foundation that you can depend on to get started. It never hurts to enter the competition with a solid skill level that will grease your way. After all, you want to sail through the learning process and get stuck in using this stuff and earn some payback.

It all started with the establishment of the Internet and the advent of the World Wide Web, which is a discussion not a thing. It's best described as an information system where resources are interconnected via addressable-activated text or images embedded within a document. Hypertext being some identified text or image when clicked in a Browser sends an address out onto the Internet to locate the desired resource like a spider, thus the term web is pretty good. Once found, the resource is returned to the sender or more specifically the sender's Internet address. All these possible addresses are interconnected in many ways and locations via electronic switching and routing.

If this sounds all a bit cryptic, it will all play out well when we get into the mechanics of how it's done. The star in this play is the Browser. Browsers house the software that speak the protocols of the Internet and the interpretation of the information received back. This is known as a query-response. So what is the message traffic like that moves to and from these addresses?

The beauty of the digital world of communication is something called data-packets. Instead of a continuous connection like the analog world of telephones a few decades ago is instead compressed nuggets of digital information that can be sent on the same wire as small discrete collections. They only take a short time for each communique. Therefore, a bunch of them can exist on the same wire in a serial fashion and be routed to lots of different users through multiple paths. So instead of just two users on a dedicated wire for their conversation, a bunch of computers can be involved in a network all sharing the same set of wires or several communication system types. Of course, these addresses have to be uniquely distinguished for the routing switches not to get confused.

Computers usually only need a small bundle of data at any one given time to get the job done. Also, the faster the data comes though, the less time needed to occupy network resources. Analog conversations would blur into gibberish if sped up super fast.

I use “wire” here to mean an interconnection, but that could be copper, optical fiber, microwave transmission or any number of technologies that move digital data. Also, note that the Internet connects software applications and not a direct communication between people. A user/person might read a screen or respond to a an LED coming on, but the actually business-end of this communication is one computer talking to another computer via the apps running at the same time. The incoming data could be converted to an analog output of sound or video but it's computer-to-computer apps doing the talking. Therefore, a massive number of packets can share the same system going in different routes all optimized for speed and then put back in order at the receiving end. The faster the better. We call this "high bandwidth" but that is a misnomer. Bandwidth is the amount of measured frequencies from one point to another. Digital bandwidth borrows this term to mean transmission speed from low to high in relative terms.

I mentioned switching earlier. Switching focuses specifically in a local setting of computers thus called a Local Area Network or LAN. Once out on the Internet, routers take on the task of finding distant LANs to dump the data, thus the term Wide Area Network or WAN. Even more specific is the sender is usually referred to as the client and the receiving end is most likely a computer set up as a server. Here again is some fuzz. Client can also refer to the app running on the user's computer. All of this activity is governed by a set of protocols so that all the participants understand how to build and write software that is compatible. Now we pick on the Browser some more.

Browsers are a complex combination of software apps or APIs and interpreters that build an image on the client’s compute screen or viewport once a transmission response is received. It does this by reading the received code and selecting a variety of built-in apps to decode what is seen. This most likely will be HTML code so a webpage can be build. HTML is a structural thing that contains the basic information to be placed on the screen. It’s the CSS code (Cascading Style Sheets) that styles the presentation to look like what we expect to see. The gorilla in the room is JavaScript that makes the page interactive in many ways from smooth scrolling to activating sections of the screen to be replaced with new data without having to reload the whole page. In fact, entire systems from the frontend to the backend can be developed from JavaScript code.

So what’s the frontend and backend. The Frontend refers to the code seen on the users or client’s computer. The focus is on design and layout so the presentation pleases the user’s eye and presents the information efficiently. The backend is the code used to run the server and control access of web pages, supporting files and database information. So web design is usually associated with the frontend process or UI(User Interface) and web development is associated with the backend process.

Now that we've explained what HTML and CSS and JavaScript are, let's get into some details. Hyperlink Text Markup Language (HTML) is a mouthful but is characterized by a set of commands hidden within a set of angle brackets collectively called tags using the greater-than and less-than symbols found on the keyboard (< and >). Between the brackets, various letters and words signal the Browser what needs to be done to build the page. For example, the <p> tag sets aside a paragraph of text. The <head> tag establishes an area where meta information can be directly communicated from the programmer to the browser like what language the page is in or where to locate additional code needed to build the page like the CSS and JS mentioned above. The <body> tag distinguishes the part of the page that can be seen on the viewport. The <head> information is not displayed. At the top of each webpage there is a declaration that the document is actually an HTML page with <!DOCTYPE html> followed by another tag, or contained within the DOCTYPE, the statement <html lang=" en"> indicating this document will be in English and starts here. There is a matching tag </html> indicating the end of the document. So there is a pattern or boilerplate of code that makes up the basic setup for any webpage that looks like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
       ...
  </head>
  <body>
       ...
  </body>
</html>
```

This template allows all sorts of information to be placed in the <head> area and the <body>. Notice the / symbol is used to close the tag. In tech-notes associated with the WolfDenElectronics website, we will get into the details of how to construct HTML. One thing to note here is this approach of encoding plain text with symbols is tags is called a markup language, meaning a plain text document can be used to invoke action by the browser. Just a file extension of .html does the trick.

CSS stands for Cascading Style Sheets, which implies there is a hierarchy to it. And there is. You apply a style though an attribute called out with either "class" or "id" precursor. An "id" is unique to the page. A "class" implies the style may be applied multiple places on the page. If you apply a style directly to an element within the page, it takes priority. The next level in the cascade is in the <head> area where you can use a <style></style> tags. Next is via a separate file whose location is found from a <link> tag in the <head>. This is used when the styles needed get lengthy and complex, which is typical.

JavaScript is a programming language that operates from a realtime interpreter found within the browser app. It has all the bells and whistles of many other languages including Object Orientated Programming. It's purpose is to manipulate the elements of the DOM (document object model), which is the tree-like structure the browser

JavaScript

Unlike most languages, JavaScript is directly attached to the HTML elements and that’s where the action is. JS code trace down through the DOM to an element called out by element name, class name, or ID name. To get the selected element into the JS code for manipulation, you assign it to a variable. For example:

```
const listItem =
document.querySelector('.list-item');
```

Notice that the query statement is a method based on “document”, which is the top level of the DOM. You can also go up a level and address the window for manipulation of the elements like scrolling and mouse positions. In this example, the “.list-item” is a class assigned to some element like a <div> tag.

Once the JS code has a grip on the <div> tag, JS can add an additional attribute to modify the styling or even add a child element to a parent. The DOM refers to parents as a tree node with attached children and siblings below in the hierarchy. You can also add an event listener for a click or mouseover, etc. to tie in user actions. Any action like this activates a callback function. This gives you a way to cause a block of instruction to commence when the code “sees” the event.

So JavaScript is like an overlord for the webpage. Normally, all the user interactions to the page would be built into the JS code. JS can add HTML elements onto the DOM without reloading the entire page, which would entail a new data query/response to the server.

Subtle things like watching user inputs into a form, for instance, can be checked and verified for correctness. If not acceptable, the input information can be ignored and an alert issued to warn the user. Only when correct data is entered will the callback function perform the next action. This helps avoid corrupted data from reaching your database or hackers from penetrating your system.

JS not only rules the DOM, it also manipulates many of the CSS properties, so transition, transform, positioning, styling, hiding/revealing, virtually any CSS functionality can be manipulated.

Network

The Browser has the ability to send what's called a HTTP Request, which is a message that conveys the details necessary to transverse the Internet's network. This is done through protocols, which are a set of rules that determine how data is transmitted between different devices in the same network. HTTP is hypertext transport protocol. This implies a bit of complexity and there is. There is a protocol stack from the Browser down to the actual wires that move the data. Each layer help form the message that will go out. The addressing is governed by IP or Internet Protocol. This establishes the sender's address and the receiving address but first a mini-message has to go out to interpret your Uniform Routing Location(URL) that is in the form of a wordy descriptor that computers can't handle like www.google.com or HarrysBarAndGrill.org. There are servers on the Internet at large that can take this gibberish and convert it into a 32-bit or 128-bit code that is registered to the entity you're trying to solicit a response from. This data is woven into the message along with whether you are requesting info or sending info, i.e., to a database. There is another term that shows up in the Request call the "host." Host is a broad term, but here we are putting into plain text the name of the computer/server or agency where your data is going to be coming from.

Typically, the messaging is a series of messages that are numbered and sent as short bursts. The receiving end sends back an acknowledgment it received it or not. If not, it's repeated. Eventually, all the pieces get there, and the server end then compiles the web page requested and sends it back to the client computer or host or a lot of other labels. Your Browser parses the incoming message from your request into all the supporting files like CSS, JS, images or other files that make up the website.

As you can see, a lot goes into requesting and receiving website information to build a DOM and render it to a user. And this is just from the data protocol point of view. We also have the electronic protocols that have to be compatible such as Ethernet to TCP to transport medium, proxy user-agents, routers, WANs and LANs just to place the one-and-zeros and clock signal on whatever is going to move your data within various junctions across the typical hardware network between you and the server. The point here is there is too much complexity to tell-all in a quaint digest format like this, but the biggies are the Browser sets up a GET (I want info from the server) or POST (I want to send info to a database) also UPDATE and DELETE message to a particular host using a particular HTTP version and lists the URL to use to collect the response back. Your request may be hustled by all sorts of proxies, routers, and groups of distributed servers or shared assets by the time a response message is constructed and sent back. Your data-packets may be framed up into many different electronic formats going from one medium to the next, but rest assured the system will keep pumping packets until your response gets back to you. This would not be complete without a mention of the various security and encryption activities that are critical to communication of important data. Also, we should note that the data format of choice these days is JSON (JS object notation), which is a fancy version of a JS Object. The difference being, JSON requires the keys to be in double quotes where a regular JS object doesn't. The data is in key/value pairs. The conversions are simple and use JSON.stringify( ) function to get your data out properly and JSON.parse( ) function on incoming data. This said, you might ponder the idea that JS could be a language that reaches further than the Browser guts and you would be right.

JavaScript in the wild is called Node.js and was created by an Open Source engineer thank goodness. Most of its assets have since been gobbled up by Microsoft. The vast majority of all the frameworks that are in use now are based on Node.js. So JS runs the Browsers and the Servers and all the data formatting in between. Everything else is a vestige of the past.

You should be able to see now why JavaScript is a hot item. Oh, by the way, Microsoft bought out GIT, the most used version control cloud-based service for most of the world's developers and integrates seamlessly into many hosting companies cloud-based services. They also bought out NPM (Node Package Manager) that is the world's largest software registry. Who's your Daddy now?