**Compare and Contrast: Server-side vs. Client-side**

With the advent of the internet and the global telemetry that supports it, companies, hobbyists, and content producers alike were given unparalleled access to a host of unique features to exploit when communicating with their consumers. Compressed video playback, private messaging forums, news aggregators, and everything in-between all answered to set of rules that at times enriched their product, and at other times limited their reach. A clear distinction between how this content was provided came in the form of client-side and server-side code, scripts, and applications.

Examining how users could interact with these differences in product deployment and development requires a close examination of the intricacies that bind them. PHP can be written as a tertiary layer for HTML, but will ultimately act as a server-side language easily compatible with networking and databases like SQL. JavaScript for example, commonly runs its scripts on a user’s computer after they have already loaded the page. Every step a web developer can take to offset the stress of each new user will help stabilize their product as a whole, and should only require their server to produce unique results when necessary. If a user opened a webpage with JavaScript code ready to display “Hello World,” the code would act just like its HTTP counter-part when called from the browser and produce its result as a “client-side” product. Certain JavaScript “engines” like SpiderMonkey, (Eich, n.d.), can interface with servers like Apache HTTP and pull data for a user to read, like databases, files, or logging process management, security, etc.; these calls are queried upon request, and act as a server-side example of server-side JavaScript code. Acting as a server-side reference, Brendan Eich provides this example from the Mozilla Developer Network, of a “click event” (usually represented as “onClick”) piped through a JavaScript engine after including the proper classes and libraries:

JSObject \*target, \*event;

JS::AutoValueArray<1> argv(cx);

/\*

\* Find event target and make event object to represent this click.

\* Pass cx to NewEventObject so JS\_NewObject can be called.

\*/

target = FindEventTargetAt(cx, global, x, y);

event = NewEventObject(cx, "click", x, y);

argv[0].setObjectOrNull(event);

/\* To emulate the DOM, you might want to try "onclick" too. \*/

ok = JS\_CallFunctionName(cx, target, "onClick", argv, &rval);

/\* Now test rval to see whether we should cancel the event. \*/

if (rval.isBoolean() && !rval.toBoolean())

CancelEvent(event);

Most objects presented to a casual user on the internet are produced in linear fashion, and a finite in their reach. Examples of these include HTML, CSS, and the aforementioned JavaScript. In concert with our previous “Hello World” demonstration, HTML and CSS can differentiate itself from the rest of its competition by being a language whose sole purpose is to interface with the user’s web browser of choice. This has the potential to drastically reduce the size of each document created, and reduce the workload of each developer. The contents of each document will only change if edited by the developer themselves, and the display of each document will always be limited to the contents the developer chooses, as seen in this rudimentary HTML “Hello World” example:

<html>

<header>

<title>This is title</title>

</header>

<body>Hello world</body>

</html>

Conversely, if a user ever wanted to ask the webpage to tell them something different, or to answer a question it might be able to find the answer to, client-side content can only reach so far. Nowadays, extensions to the browser can extend the reach of client-side scripts through constant updates and complex networking under the hood, but the results a user sees are always dependent on what their extension can find after calling back to a server for confirmation. Carey Wodehouse, a freelance content marketer and writer from Richmond, VA, expands upon this idea in an article titled “Front-End Web Development: Client-Side Scripting & User Experience,” for UpWork.com. She writes, “All websites run on three components: the server, the database, and the client. The client is simply the browser a person is using to view a site, and it’s where client-side technology is unpacked and processed. The server is at a remote location anywhere in the world—housing data, running a site’s back-end architecture, processing requests, and sending pages to the browser. The client is anywhere your users are viewing your site: mobile devices, laptops, or desktop computers. Server-side scripting is executed by a web server; client-side scripting is executed by a browser,” (Wodehouse, n.d.). If we are to fully utilize the tools at a user’s disposal, then a combination of *both* client-side and server-side scripting can take advantage of the full scope of the industry. PHP uses a server to produce its content in HTML upon request, and uses that same server to call back for information on users, tables, metrics, and other result searched by the user. Certain HTML5 components offer the same results at a reduced strain on the server itself.

As can be expected, web development and other networking tools are fueled by an innovative spirit and an interest in working smart, not hard. Constantly streamlining content means cutting back on as much fat as possible, and newly introduced languages help offer the best of both client-side, and server-side applications. JQuery is a JavaScript library that expands the reach of JavaScript, and ‎EaselJS takes it a step further with a focus on HTML5 integration. No matter the language however, users will continue to expect a faster, more responsive product, and as internet service providers become faster and faster, the potential for a fully-realized “cloud-based” computing architecture comes closer and closer to reality.

Works Cited

Eich, Brendan. "How to Embed the JavaScript Engine." Mozilla Developer Network. Mozilla, 17 May 2016. Web.

Wodehouse, Carey. "Front-End Web Development: Client-Side Scripting & User Experience." UpWork. UpWork, Inc., n.d. Web.

Spainhour, Stephen. "WebMaster in a Nutshell." O'Reily, Oct. 1996. Web.

Flannagan. David. " JavaScript: The Definitive Guide.” O'Reily, Jan. 1997. Web.