Web Development: Executive Briefing

by Brice Wilson

Start Course

Bookmark Add to Channel Download Course

Table of contents Description **Transcript** Exercise files Discussion Related

What Is Web Development?

Web Development Overview

Web development is creating websites or applications that are meant to be viewed or used from within a web browser. Web browsers exist on just about every computing platform these days. The most popular browsers are Internet Explorer for Microsoft, as well as its successor named Edge, Google's Chrome browser, and Mozilla's Firefox. Browsers are designed to understand a standard set of technologies. Ideally this means that developers can program to those standards and be confident that their applications will run anywhere for any user with a browser. This isn't always the case, but it's certainly more true than it used to be. Building applications on top of the standards supported by web browsers is so popular because it's easy to distribute those applications to anyone in the world with computer, a browser, and a connection to the internet. When you hear people talk about web development, you'll often hear them refer to websites and web applications. These two terms are often used interchangeably, but they do hint at two somewhat different types of web development that exist. Websites are generally just a tool for distributing information. They're often just a collection of static web pages that present information about a specific topic. They'll have links to the various pages on the site and may be links to other sites on the intranet, but beyond the links on the site, there isn't much interactivity. An example of a static website might be a blog you like to read or a small local business site that just has a few pages describing the services they offer. The goal is really just to convey static information. Web applications are much more interactive. They can be programmed to do lots of different things, but they are particularly good at collecting data from users, storing it somewhere, and then presenting it again when needed. For example, consider a hotel reservation application. It takes information about the dates of your stay and the type of room you'd like and stores it in a database somewhere. That data is then accessible to the hotel staff, as well as to you if you return to the application later to confirm your reservation. Applications can also convey information like a website, but they usually provide a lot more interactivity and allow you to perform some set of tasks. Building a website or a web application

Web Application Architecture

Building a website or a web application really involves two distinct types of development--server-side development and client-side development. I'll talk more about each of those later. But for now, I just want you to understand the general architecture. In almost all cases, an active network connection between the server and client is required in order to use a web application. Typically when a user clicks on a link or performs some other action in a web browser, a request is sent to a server somewhere on the internet. The server will then execute some code or locate a requested file and deliver it over the internet back to the browser. Client-side code may then execute inside the browser to further process the data sent from the server. Without an active connection to the Internet, none of this communication is possible. I mentioned earlier that web development is performed against a set of standards that are supported by all web browsers. That also applies to the network connection between browsers and servers. They communicate using a network protocol known as HTTP, which stands for Hypertext Transfer Protocol. The HTTP protocol can also be encrypted and is then referred to as HTTPS, the S being short for Secure. You have undoubtedly seen HTTP and HTTPS as the start of a website address. It's really there as an instruction to the browser. It tells the browser to use the HTTP protocol to communicate with the specified website. Most browsers support a handful of other protocols, but HTTP and HTTPS are far and away the most commonly used.

Progressive Web Apps

Not all web development is done with a desktop web browser in mind. Since the earliest days of the modern smart phone, there have been attempts to make web applications look and perform like native mobile applications. The techniques used to do this have evolved over the years, but the current best practices have been collected into a set of techniques to deliver what is known as a progressive web app, a PWA for short. They use some advanced and emerging web development techniques to give users an experience similar to the native apps they've become accustomed to. PWAs load faster than the typical web application. They can also be installed on a phone like a native app, and often have other features not typically associated with traditional web apps like storing data on the client and receiving push notifications. They're certainly worth considering if you're interesting in leveraging your existing web development teams to also deliver a nice mobile experience for your users. And speaking of teams, in the next module, I'll talk about the skills required on a typical web development team. Stay tuned.

Team Structure

Client-side Developers

In this module, I'll go over the skills required on a typical web development team and how they may or may not neatly map to any given group of developers. Let's start by talking about client-side developers. These are the people that write the code that will execute inside the user's web browser. The browser presents the user interface of an application, so good client-side developers often have user interface and user experience design skill. These are often abbreviated UI/UX, and people with those skills are often referred to as UI/UX developers. They use Hypertext Markup Language, HTML, to lay out the structure of a webpage and cascading stylesheets, CSS for short, to apply visual styling to those pages. I'll talk more about HTML and CSS later in the course. Client-side developers are also usually very familiar with the JavaScript programming language. JavaScript is a full-featured programming language that is supported by all of the major web browsers. The popularity and capability of JavaScript has exploded in recent years, which means that many modern web applications have as much or more client-side code as they do server-side code. That was not the case in the early days of web development. Client-side development has even grown to the point that it's not uncommon for there to be some specialization on larger teams. There may be developers that focus more on HTML, CSS, and the user experience, and others that focus more on writing JavaScript. However, you can't build a web application with a team full of client-side developers. You also need some server-side developers.

Server-side Developers

Server-side developers, as you probably guessed, focus on writing the code that executes on the server. The skills required to be a server-side developer are much less specific than those for client-side developers since server-side developers aren't bound by the specific technologies supported by web browser. There's more flexibility when choosing a programming language and application development framework, but it's still necessary for server-side developers to be proficient in at least one of them. I'll talk more about some of those options that exist later in the course. Server-side developers generally spend less time worrying about the user interface and more time gathering, processing, and delivering the data and resources needed on the client. This often includes things like gathering data from an internal company database, making calls to third-party services, or just implementing proprietary business rules and calculations before sending the results down to the client. Because of this, it's often important for server-side developers to have experience with relational databases and integrating with disparate systems. It's true that just about every web application will require both server- and client-side development skills, but it's increasingly the case that members of many teams have skills that span those required for the server and client. I'll talk about that development trend next.

Full-stack Developers

Over the years, the lines have blurred between the skills required for client-side and server-side development. The rise of JavaScript has many client-side developers processing more data and doing less design work. The same need for more data processing on the client has led to more server-side developers working on the client and picking up some UI/UX skills along the way. This has all given rise to what is now known as the full stack developer. A typical web development stack might include HTML, CSS, and JavaScript development on the client, a different programming language on the server, and likely some database-specific code to interact with an underlying database management system. A full stack developer is someone that's proficient at working in all of these areas. This has some obvious benefits. Developers familiar with the whole stack don't have to wait on the client-side developer to come back from vacation to troubleshoot a JavaScript bug. They also generally have a more holistic view of the application, and this can lead to better system design decisions. One potential drawback to filling the team with full stack developers, though, is that it's difficult to maintain a deep knowledge of all of the technology used in a typical web stack. Rarely does a generalist have the deep knowledge of a specialist.

Additional Skills That May Be Required

In addition to client-side developers, server-side developers, and maybe the occasional full stack developer, there are a few additional skills that will likely be required to successfully develop and deploy a production web application. These include system administration, database administration, and network administration. The specific tasks performed might include configuring and securing the servers that host the web app, creating and tuning the database, and configuring firewalls and routers to allow all of the appropriate network traffic through. In larger organizations, there may be separate people that specialize in each of these things. In very small organizations, the developers may have to pitch in to handle them. These are generally peripheral tasks when you think about the development of a medium to large web application, but that does not mean they're not important. It's always better to know ahead of time that you'll need those skills and have people with them ready to help when it's time to deploy that shiny new application.

Client-side Development

Coding for a Browser

In this module, I'm going to cover client-side development and the most popular technologies used to build applications that run in a web browser. I mentioned earlier that one of the reasons for the popularity of web development is that developers can code to the standards supported by web browsers and trust that their code will run on any computer in the world that has a browser. This is true, but the reality is not quite so idyllic. The reality is that all of the browser makers aim to support web standards, but they're very competitive and are always working to make their browsers faster and more user-friendly. The result is that there are occasionally small bits of code that work perfectly well in one browser and either generate an error or maybe just look a little different when rendered in another browser. The browser renders the user interface of a web application, so even slight differences in how a particular browser displays a button or positions a user input form can be very noticeable and distracting. The likelihood of running into one of these compatibility issues is not near as great as it was several years ago, but it's still something developers need to consider. Unless you know that you're developing an internal corporate application, and you know that all of your users will be using a particular browser, then part of web development really needs to be testing your application on multiple browsers so you can be sure all users will have the experience you expect.

HTML and CSS

HTML and CSS are the two technologies that most directly control the visual appearance of a web application. HTML stands for Hypertext Markup Language, and it's used to define the structure of a web page. Developers use HTML to do things like add a new paragraph of text to a page, add the input boxes for a data entry form, or include a button the user can click to have their data saved to the server. There are a few pieces of the HTML specification that let developers control the visual appearance of the elements being added to a page, but for the most part HTML is about defining what goes on a web page. How those elements appear on the page is really the domain of cascading stylesheets, more commonly known as CSS. CSS allows developers to apply styling to the page elements they defined in the HTML. That styling could include things like particular fonts or text formatting, the colors used on a page, or how HTML elements are positioned relative to one another. It's important that developers define the HTML for a page separately from the CSS. It's often helpful to users to adjust the visual appearance of a page depending on the type of device a user is using. If they're using a desktop computer with a large monitor, then an application might detect that and load a CSS file that formats the HTML with lots of menus and navigation items that might not fit well on a smaller display. If the application detects that the user is using the web browser on their phone, then a different CSS file might be applied that hides some of the menus and reformats the content into a form that's easily scrollable on a small screen. It's still the same HTML code, but the CSS is used to detect and present it in the best format for the device being used.

JavaScript and JSON

HTML and CSS are great at structuring and presenting web pages, but adding lots of interactive functionality requires a full-featured programming language. In web browsers, that language is JavaScript. The official name of JavaScript is ECMAScript, named after the organization that standardizes the language. However, JavaScript is the name you'll hear most often. The earliest versions of JavaScript were really just used to add small bits of flair to web pages. Over the years, the language evolved, and developers started doing more and more with it. It's now not uncommon for a web application to include as much client-side JavaScript code as it does server-side code. Writing JavaScript code instead of server-side code has a couple of particular advantages. First, it reduces the processing load on the servers. Code that used to execute and take up CPU time on the server can now execute in the user's browser. Second, because the code is executing in a browser, it improves the overall user experience because the application will generally be more responsive to the actions taken by the user. And when data needs to be sent to and from the server, JavaScript can use a special data format known as JavaScript Object Notation, JSON for short. It's a very human readable text description of data that JavaScript can easily process.

AJAX Requests

Until the mid-2000s, if the user of a web application took some action that required a partial update to a web page or required that some data be retrieved from the server, it usually meant that a request had to be sent to the server, and an entirely new web page had to be sent back down to the browser. This was true even if it was only a very small portion of the page that was different from the original. Eventually a set of technologies, collectively known as AJAX was added to JavaScript and supported by browsers. AJAX stands for Asynchronous JavaScript and XML. Let's break those terms down so you can understand how this fundamentally changed client-side web development. AJAX requests are network requests made to the server asynchronously, which means that they're really made in the background so that they don't disturb or delay the other processing that may be happening in the user interface inside the browser. The alternative is to make requests synchronously. The problem with synchronous requests is that nothing else can happen in the browser while it waits for the request to go to the server, perform some process, and then return to the client. This appears to the user as if the application or their browser is frozen or locked up. I'm sure you've seen applications that exhibit this behavior, and you know how frustrating it can be to use them. By performing tasks asynchronously in the background, the user interface can remain responsive and continue to update based on user actions while it waits on the response from the server. I told you that AJAX stands for Asynchronous JavaScript and XML. The XML refers to a data exchange format that has largely been replaced with JSON in modern web applications. With the ability to make requests to the server asynchronously, there was suddenly lots of potential for building web applications that operated much more efficiently and only updated portions of a page at a time. That ultimately led to the development of modern client-side application frameworks, which I'll talk about next.

Libraries and Frameworks

As the capabilities of JavaScript grew, and more developers started to take advantage of AJAX requests, it became possible to build many more application features in the client-side apps than ever before. Client applications began to resemble server applications in terms of complexity and the quantity of code being written. That level of complexity led to a need for larger application building blocks that already had in place some of the most common bits of code that many applications needed and that would speed the development of individual features. The first really popular tool that began to fill that need was a library named jQuery. Among other things, it includes lots of built-in functions that make it easy for developers to make AJAX requests and then update particular pieces of pages with the results that come back from the server. I refer to jQuery as a library since it's really a collection of helpful functions developers can easily use with just about any client-side app they need to build. The huge success of jQuery eventually led to the development of what I'll call client-side application frameworks. Rather than just being a collection of utility functions, frameworks provide much of the scaffolding needed for medium- to large-size modern browser applications. Some of the most popular frameworks are Angular, which was developed by Google, React, which was developed by Facebook, and Vue, which was originally developed by a former Google employee and Angular user. These frameworks let developers build what has become known as a SPA. SPA stands for Single Page Application. Not all SPAs are literally a single page, but the general idea is that the client-side app can load a single page, and then all of the functionality inside the app consists of small partial updates to that page. Client-side frameworks enable this type of application by making heavy use of AJAX requests. The result is often a very fast and responsive application that feels less like a traditional web app and more like a native application. Each of the frameworks I mentioned has a bit of a learning curve, but once developers become familiar with one of them, they can usually get big full-featured applications off the ground much quicker than if they had to code everything from scratch.

Server-side Development

Role of the Server in Modern Web Applications

Despite the huge growth in the popularity of client-side development in recent years, server-side development is still a very important part of just about every web application. There are really three main roles the server plays in the typical web app. The first is that it runs some web server software, and this really just means it's running an application that can listen for and respond to requests made with the HTTP protocol I discussed earlier in the course. The second job of the server is to deliver to the client all of the HTML, CSS, and JavaScript needed to run the browser application. In some, but certainly not all cases, that client-side content will reside in files on the server. When a user visits an app with their browser, the server delivers those files so the user interface can immediately be shown, and the app start receiving input. The server may deliver other client-side content over time, but there is usually a minimum set of content that must be sent to initially get the app running in the browser. The third job performed by the server is to execute custom code specific to your particular application. This is the largest of the three jobs the server normally performs and can obviously take many different forms. Some common tasks to perform on the server include saving and retrieving data to and from a relational database, retrieving data from other internal corporate applications like HR and financial systems, and connecting the third-party applications or services that contain other information needed by your app. In addition to gathering and delivering data to the client, it's also common for the server-side code to implement business rules that apply to your application. It can be beneficial to implement business rules on the server in case you decide to use the same server code for multiple client applications. They can then both take advantage of one implementation of the business rules.

Programming Languages and Frameworks

Unlike client-side developers, server-side developers aren't limited to using the technology supported by web browsers. Server-side programming languages and frameworks need to be able to communicate over HTTP and ideally be able to send and receive JSON data. And there are lots of languages that can do those things. In addition to the programming language itself, the best options also include a framework that already has much of the infrastructure in place for communicating with the client over HTTP. One popular choice is Microsoft's ASP. NET Framework. It's built on top of their popular. NET application development framework and works with several programming languages including C# and VB. NET. Java is another popular choice. It's a full-featured, general purpose programming language, and there are lots of web frameworks you can choose to use with it In recent years, Node. js has also become a popular tool for building server-side applications. Node applications are written in JavaScript, so it has the advantage of letting your team use a single programming language on the client and server. Python and PHP are also popular server-side programming languages, but don't assume that your only choices are those I've mentioned here. There are lots of options, and choosing a server-side language and framework can have as much to do with the existing skills and experience on your team, as it does with the technical features offered by the different tools.

Execution Environments

Server-side code obviously requires a server, and that comes with its own set of choices when deciding how to architect your web application. Windows and Linux are easily the two most popular web server operating systems. But keep in mind that Linux comes in several popular distributions from a variety of organizations. Which server operating system you choose will probably depend on the existing skills and experience on your team, as well as the technical standards that may already be in place at your company. In addition to choosing an operating system, you must also choose where to host your application. If you work in a large organization, you may already have in place a datacenter with the infrastructure necessary to support and maintain your application. However, it's become increasingly popular for organizations of all sizes to host web applications with one of the large cloud providers. Microsoft Azure, Google Cloud, and Amazon Web Services are all popular cloud hosting providers. There's a common joke that the cloud is just a fancy name for someone else's computer. That's good for a laugh, but if you assume that oversimplification bears any resemblance to modern cloud architecture, then you're likely going to miss out on some significant benefits offered by these services. Cloud architecture can easily be the topic for an entire course, and there are lots of benefits to hosting with a cloud provider, but the one I'll mention here is that you can usually very easily scale out an application to multiple servers in the cloud. This can be handy if you suddenly have a spike in traffic or if there are certain times of the day or week when you need more capacity. Implementing something like that on your own servers often requires a lot more effort and expense.

Final Thoughts

Web development these days involves more technologies and a greater variety of skills than ever before. However, it's also more possible to deliver fast, beautiful apps with more capabilities than ever before. I hope this course has given you a better understanding of how to build modern web applications. When you're ready to dive deeper, Pluralsight has lots of courses available on all of the topics I've covered. I hope you'll keep learning. Thanks for watching, and good luck on your next project. We hope you enjoyed this course. If you're interested in more content for technical leaders, content we keep short and focused with the up-to-date information you need to be informed and make decisions but without getting buried in the details, find other courses like this at plrsig. ht/exec.

Course author

Brice Wilson

Brice Wilson has been a professional developer for over 20 years and has used many tools and programming languages during that time. His current interests are centered on web services, single-page...

Course info

Level	Beginner
Rating	★★★★☆ (56)
My rating	★★★★★
Duration	0h 30m
Released	31 Jul 2018

Share course