Front End Web Development Career Kickstart

by Justin Schwartzenberger

The Web Development Industry

Introduction

How's it? I'm Justin Schwartzenberger, and I am stoked to be teaching you this course, Front End Web Development Career Kickstart. Building a career in front-end web development involves way more than just knowing the X's and O's of HTML, CSS, and JavaScript. Where a web development boot camp or training program will kick start our coding skills for the front end, this course will get us dialed in with everything we need to know and be aware of to take that focused training and turn it into a career. This course is going to be our guide, our blueprint to not only launching a career in front-end web development, but how to grow and cultivate that career once we begin it. If a course on learning front-end web development with HTML, CSS, and JavaScript is, let's say, the way to learn to drive a car, this course is the one that teaches what needs to be known to drive that car in the city. We will explore the makeup of the web landscape and get acclimated to the terms and technologies that drives a web page or application. Stepping through the full stack, we will cover the roles of servers and clients, the transport that connects the two, and the addresses that help with navigation. The web is full of different languages. We will drill into the front-end trifecta of HTML, CSS, and JavaScript, as well as those foreign languages from the server side. Our journey through the web landscape will conclude with a focus on browsers, a front-end web developer's canvas. Then we will uncover the details as to what is involved in doing front-end development. Learning paths will be discussed for mastering HTML, CSS, and JavaScript with an emphasis on the latest versions of each. We will open the toolbox and take a look at what we have at our disposal to craft front-end websites. That exploration will start with a look at browser developer tools and how we will use those in our daily workflow. And then we will need to discuss our primary tool, either and advanced text editor or an integrated development environment, or IDE. There are many choices out there. We will check out a few and learn what they bring to the table and how they not only aid in the front-end web development process, but why it is necessary that we understand and are aware of IDE options on a regular basis. We will finish up with a discussion about version control, making sure we learn the core concepts of it and how to use it. With all the concepts, tooling, and workflow covered, we will turn our attention to what it takes to get prepared and go on an interview for a front-end web development job position. Finally, we will discuss several things we can do to grow our front-end web development career from the moment we begin it, things like self-education, networking and mentorships, always keeping our finger on the pulse of the industry, and continuous tinkering with new projects and technologies. Links to resources that cover these terms and topics can be found in the Exercise Files. But to begin the course, we will take a trip back in time and revisit the birth of front-end web development and the history surrounding the industry and this career path that we are pursuing.

History Part I - Where It Began

Part 1 of our history lesson: Where the web development industry began. In professional sports, it is referred to as being a student of the game. To know where you're going to take it, you need to know where it's been. The web development industry is young relative to other industries, yet it has matured at a much more rapid pace than many others. Knowing the back story not only helps us to understand why architecture advancements and decisions are made today, but also provides opportunities for discussion and resonation with web development veterans whom we will be working with as we enter the industry. Alright, let's fire up the DeLorean or Tardis or hot tub, whatever our time machine of choice is and head back in time back when the web itself was young and very few businesses even understood how it worked and what purpose it served, let alone had any inkling that they needed a presence online. Front-end web development wasn't anywhere close to being an actual career. People were starting to learn to build for the web, but they weren't really getting paid much, if anything, to do so. The market just wasn't there yet. Part of this was due to the lack of understanding of the power of the web and what it could become. Businesses and individuals needed to be educated on it. The other part was that there wasn't a good story for the material to teach them about it. The web itself was raw and untapped. As far as consuming content from the web, end users didn't have many great options. The story to get online involved signing up with an internet service provider, or ISP for short, receiving an install disk that either had Netscape Navigator or an application for configuring Internet Explorer, and waiting for a modem to log in each time you wanted to do anything online. A user would need to be highly tech savvy to even consider trying to use some other browser. Heck, most of these browser installs or configurations from these ISPs came with some customization or toolbar that changed the operation of the browser to a more proprietary experience tailored around that ISP's branding and other services. Browser rendering engines were raw as well. HTML and CSS specs were in their infancy. JavaScript was, well, it was this frightening black hole that most were afraid to explore for fear of unleashing some great evil that would wreak havoc and destroy everything on the page and possibly the end-users machine. And tables, tables were the backbone of every design. Yep, table. Tr's, td, even tbody's were the kings of the court. Grid layouts, well they were table grids minus the flow part. Building for the web at that point in history meant playing around in these gated communities and adhering to their rules. It was rough. Domain name registration was expensive. Competition in the domain registrar space was non-existent, so rates were high. To get web site files deployed to a server, FTP was used. Yeah, FTP. Developers would have to manually push files to a web server. Version control of project files and resources basically consisted of the fact that a developer would have a copy of the files on their machine and a copy of the web server. From the front-end development side of things, the creation-side story was just as raw. There were no advanced text editors or integrated development environments. There were basic text editors like Notepad and VI. A few players started to come into the mix with the intent to bring some drag-and-drop building experiences, applications like FrontPage and Dreamweaver. These were a start in the right direction, a start for the front-end web development tool industry. In the next clip we will cover the rise and demand for a web presence and what it meant for the web development industry.

History Part II - Everyone Needs a Site

Part 2 of our history lesson: Everyone needs a site. Up to this point when asked, what do you do for a living, one could really not tell someone that they did web development and expect to have any understanding from the other party as to what that meant. Then, what felt like overnight, businesses became internet aware. More and more people were online and learning the ropes of the internet, seeing it's potential. Businesses started to buy into the need to have an online presence. Thus began the rise of the web development shop and the first real declaration of a career based around designing and developing websites. As web shops opened up and started to find their groove, turning out site after site, they started to work on frameworks and architectures that they could reuse across projects. Content management systems, or CMS for short, were built that allowed for visual customization with dynamic content added via web admin tools after the site was launched. Use of the web and websites was on a massive rise. Sites were starting to feel the effects of heavy visitor traffic loads. A lot was learned at this time about web development in general and things like web architecture patterns and client and server-side performance that shaped the way we approach web development today. During this period, a career as a full stack web developer became a thing. Companies started hiring specifically for that focus, not just software engineers. Front-end specialization wasn't quite a thing yet, mostly due to the fact that the design of the web and languages used at that time involved writing a mix of server and client code delivered by the server at each action request. Web APIs had yet to become commonplace. With the rise of web shops came the market for companies to build products and solutions to meet the needs of web shops. The tooling for doing web development started to mature. IDEs started to turn up that focused on writing code for the web, with most centering on a particular language. Other web browsers started to come to market. Google launched its Chrome browser, and Mozilla came to life from the ashes of Netscape Navigator and released Firefox. Other players came to the field as well, like Apple's Safari and the Opera browser. Web developers quickly discovered that markup, style sheets, and scripting wasn't a write-once, run-everywhere scenario. In fact, there were little in the way of web standards, and browser vendors were so caught up in market competition that web developers faced many challenges when it came to creating and maintaining a website codebase that worked the way they intended on multiple browsers. Web design and user-interface design started to ramp up at this time as well. People started to explore how to make the web beautiful and more useful. Graphic design started to take a front seat, and with that came the need to understand how to translate mockups created in graphic programs like Photoshop into markup and style sheets for the web browser to render. As web design matured and advanced, the task of converting it into markup for browsers required a stronger understanding of HTML and CSS and its capabilities and limitations, as well as discovering and unlocking new combinations and approaches to solving design challenges within the render rules of browsers. This really set the stage for the need for front-end web development specialization and what would become the possibility of a career in front-end web development. In the next clip, we will discuss the rise of the mobile platform and how it changed the approach to web development.

History Part III - Mobile Support

Part 3 of our history lesson: Mobile support. With the dawn of the age of smartphones, internet users began to carry their browsers in their pockets, and all those businesses that got their online presence going quickly discovered that their sites don't look so hot on a mobile device. Some failed to even function correctly. There was a knee-jerk reaction to the problem that saw web shops building mobile versions of sites to fill the need for a different experience on a smaller screen. All of the sudden web developers found themselves needing to learn about browser rendering and functionality on mobile devices and determine ways to port over existing sites and web apps into mobile versions. And for a while there was a trend of having a www site and an m. site where www. thesite. com was the desktop version, and when visitors hit the site from a mobile device, code would detect that the client requesting the content had a mobile-like user agent string and would redirect the request to the domain m. thesite. com. But as developers worked to build out these mobile-site versions, they began to identify ways to craft HTML and CSS that would scale and change based on changes in the viewport size. Responsive web design was born, and techniques were established, documented, and shared as to how to craft one code base for a website that would look great on a desktop browser and would adjust the content and layout flow to look great on a smaller device all based on CSS and things like media queries and feature detection. So part of this web development achievement came from devs engineering new solutions, and part came from the standardization in specs for HTML and CSS and browser adoption of these and of new features. At this time, native mobile apps were born, took off at a rapid pace, and never looked back. These native apps were not initially constructed on front-end languages, so at that time they didn't provide a potential career path for those working in the web development industry. What they did, however, is drive the dependency of being online and always connected through the roof. And as people started to consume these apps, they wanted more. They wanted apps for all their things, including those things that they were previously consuming via websites. Social media, sites and services like Facebook and Twitter, people wanted a mobile-app version of those, and they got them. And when they did, the awareness of those sites grew tremendously. Then those social media sites started penetrating the mainstream. TV shows, ads for companies, physical products, they all started to promote Twitter handles and Facebook account URLs. The web, and the understanding of how to use it, as well as the expectations based on existing sites and patterns, had become commonplace. The average consumer of the web didn't know what was involved in making it work, but they were fully aware of it working and it being an everyday thing. So when someone would ask a web developer, what do you do, and they responded, I create websites and web apps, the response the developer got was no longer a blank stare. Instead, it was met with excitement and intrigue and even a bit of awe. The recognition and concrete establishment of front- end web development as a serious career was finally a reality. In the next clip, we will go back to the future and talk about what front-end web development is today.

Where We Are Today?

Ending our history lesson: Where we are today. Man, today is a great time for front-end web development. The maturity of the toolset around building front-end code is at an all-time high. We have great desktop applications for writing HTML, CSS, and JavaScript. Browser web dev tools are releasing with new features at a rapid cadence. We have web applications like Plunker, JSFiddle, and Codepen that allow us to prototype and test front-end code and even create and deploy full-site code from a browser. Companies are being launched that have a primary focus on building tools and services purely for front-end web development support. And we have browser vendors adopting web standards. The major players, Chrome, Firefox, and Microsoft Edge, all support an auto-update model where the latest features are automatically rolled out to the end user without them needing to manually install the latest version. So as new standards for features in HTML5 and CSS3 get approved, these vendors can deliver support for those right away to all the install base of those browsers. As front-end web developers, we get the opportunity to build against those new features early and often, a far cry from the days of old where future features were the things of dreams locked away in some vault of hopes only to be discussed with what-ifs. The web development industry has matured. No longer is it trying to find its way in the World Wide Web. Patterns, practices, codes, and services all exist for the industry. There are years of experience gained around the landscape of the web. This experience is readily available in the form of blog posts and co-documentation and training videos. Not a month goes by without some sort of conference based around front-end web development or some front-end web framework. The fringe of the web is being explored, and boundaries are being pushed. All the common challenges have been solved. Web developers are free to focus on cranking it up to 11. Problems are being solved with new approaches based on years of better understanding and experimentation around web technologies, and the range of career opportunities is vast. We can build a career around front-end web development, can become a full stack specialist, or focus purely on mobile. We can take on building and maintaining a front-end web framework library, joining a team working on something like jQuery, Angular, or Auralia, or we could build a career around plugin development either for an existing framework or even a browser. We can even start using our front-end skills to build native apps, both for mobile hardware and for desktops. Front-end web development as a career is more popular than it has ever been, and the opportunity to not only be a part of it, but to advance it and learn and thrive, well, it's all on the table. Where we are today, this is the best time to pull up a chair and dig in. With our history lesson complete, the next clip will cover the many different career paths available to a front-end web developer.

Career Paths

Now that front-end web development is a thriving industry profession, there are many variations of career paths within it. These variations center on different specializations. As we learned in the history clips, the full-stack web developer is an extremely common path, which has an equal balance between front-end development, building stuff for the client side, and backend development building for the server. Sometimes, depending on the technology and frameworks used, project or company position may require someone who is a front-end web development specialist to be working in more of a full-stack arena. If, for example, we find ourselves working on a PHP project, we as front-end web developers would be building HTML mixed with PHP code to include data. It would be front-end stuff, but would be mixed with server-side code that helps aid in rendering the front-end code and content. But front-end specialization does exist and is growing each day. With an increasing web architecture of backing API services and endpoints, the full stack is gaining more separation with the goal of being able to layer different clients to consume data from servers. Within this landscape, there is the opportunity for front-end developers that live in an HTML, CSS, and JavaScript world where client-side websites and applications are designed, built, and animated with HTML and CSS, and JavaScript is used to communicate with API endpoints and handle all the user interaction. This abstraction of the back end from the front end allows for other front-end application solutions, like mobile apps. While most mobile apps are built using native code, there are avenues of creating mobile apps out of front-end web code. These consist of apps or services designed to package up an HTML, CSS, and JavaScript app within another native app so they can run on a mobile or tablet device. These apps and services also provide hooks into the native functions and services, allowing a front-end developer to tap into the use of hardware like the camera or GPS on a device. The mobile space is one that is gaining traction for front-end web developers and offers a lot of career opportunities. A plus with this mobile-web-development approach is the idea that one app code can be built and can be used across different delivery avenues with very minor differences. Where to work? So where would we work? Well, there's employment in company, freelance or contract work, or we could take on building our own web-based product or service. Each of these has some additional learning path needed. If we want to pursue a position at a company, we'll be facing an interview process to land a job, and our positioning will largely be dictated by our project experience and landscape knowledge we can show. A front-end web developer job position will involve crafting HTML and CSS, typically from design mockups, to implement new features to existing web apps and sites and writing JavaScript to implement user-experience interaction and data persistence. It will also involve working with company-selected tools and software, most likely a single IDE app and a source-control solution or version control. Some of the benefits to a company position is the ability to work within a team environment and the opportunity to learn and grow our skillset from that environment. Training, mentorship, and constant visibility of other developer's code all provide several paths for personal career growth that can lead to career advancement. Becoming a freelance front-end web developer is something usually approached by developers who have spent some time in the industry, experienced with multiple projects, the multitude of programming and design challenges that one would have to come across and have to solve. And the experience of working through project feature timelines and planning are all skillsets that have high translation value to taking on freelance contract work. But that doesn't mean that freelance work is not possible for someone just starting in a front-end web-development industry. In fact, with the state of the web today, there is a wide variety of contract needs available out there that range from projects or tasks that don't require years of technical experience to those that do. A freelance role will have us working with existing projects and version control around them, but we may have a need to run our own version control and tooling as well. A third career option for front-end web development is to take on building our own application or site idea. This approach will most likely require us to work more of the full stack, but there are more and more services and solutions coming to market to handle server- side stuff that is making it easier for us to stay in the front-end landscape and focus our expertise there and turn out sites and applications that are serviceable. Building our own application or site will basically have us doing all of our own business stuff. We would be handling not only front-end development, but figuring out tooling and version control, project management, and all the stuff that accompanies having a business.

Summary

In this first module, we took a trip back in time and explored the history of web development and the evolution of front-end web development specialization. We got a feel for how things used to be and the origin of current patterns, frameworks, and architectures. Learning about the history of the industry aids us in connecting with veterans in the industry and allows us to build a stronger understanding of how and why frameworks are designed the way they are and what problems and challenges have been solved with existing tooling and software geared toward the web development industry. We also learned of some career-path options available to us in the industry. From working the full stack to focusing on front-end web development, we can look to do freelance work or take on building a site or product, or we could find a job in an existing company or startup. Taking a position at a company is the most common and highly beneficial start to a career in front-end web development. Instead of getting thrown into the fire when taking on freelance work, a position provides us an on-boarding opportunity, a chance to work our way into the nuances and workflow of everyday development and to interact with and learn from other experienced developers that have made the same journey we are embarking on. But to go on that journey, we need to come prepared. In the next module, The Web Landscape for a Developer, we will take a look at what we need to understand about the website experience as a whole, beyond just front- end web development, to be ready to work professionally in the industry.

The Web Landscape for a Developer

Introduction

Front-end web development is not just about knowing how to code and implement design. It is important to understand the landscape of the web. In this module, The Web Landscape for a Developer, we will go over what we need to know about that landscape and how we can go about learning that information. First off, we need to understand both sides of the web game, the client side, the server side, and the transport of data between the two. In this module, we will dig into the details of each, beginning with the client side and cover the role of browsers to receive and render the content of the web. A discussion of the server side of things will follow on its heels covering how a web page is served up and the roles that the server plays. The last piece of the puzzle will be the transport layer. We will go over HTTP and what it means, the concept of stateless requests, and do a brief run over the role of DNS and web address. Then we will go over the languages of the web and discuss what domains they live in. We will cover the front-end trifecta, HTML, CSS, and JavaScript, as well as server-side languages like C#, PHP, Ruby, etc. Finally, we will cap this module off with a more detailed discussion of browsers. We will get to know the major players and uncover their rendering differences. We will also take a look at the role of browser developer tools. At the end of each clip, we will receive guidance as to how to further our learning on each topic. By the end of this module, we will have a solid understanding of what makes up the web landscape as it pertains to our goal of becoming a career front-end web developer.

Client Side

As a front-end web developer, we will be spending the majority of our time building for the client side of the landscape. This primarily means HTML, CSS, and JavaScript that will be parsed and executed by the browser. So when we hear the term client side, what exactly is that referring to? Well, it typically starts with the end user's browser that they are using to consume web pages. The client side is made up of a series of events and experiences that occur within the browser when it sends off a request for some content and what it does with that response it receives. The things we need to understand when it comes to the landscape of the client side are the different ways website code is built and delivered to the client. From server-side generated to static site files and packaged up web code delivered to mobile devices, it is important that we are familiar with options. When we build for the client side, we are building the instructions for the browser that tells it how to render content and provide it with the code that tells it how to handle events. The client side will ask the server side for some content or let it know it should do some action, and then typically it will receive some response, and that is about the extent of its knowledge of the server side. While the client and server side work together to make up a website or web application, they are viewed as separate pieces to the puzzle. Now, there is some overlap. The code for the client side, the HTML, CSS, and JavaScript need to get delivered to the client, and sometimes this code needs to be dynamically constructed based on the request settings for it, so it is common to have server-side code that constructs front-end or client-side code. Yet, it is also becoming common to have HTML, CSS, and JavaScript constructed without a server-side language and just served up as resources from a web server. This type of client-side architecture involves using JavaScript to communicate with APIs to pull and push data. This separation of concerns and decoupling of the client side from the server side opens up the opportunity for client-side code to be bundled up and make its way into native apps, both on mobile and tablet devices. These recent expansions of new architecture have also provided a growth and potential for web developers to focus and specialize as front-end web developers. Whereas before, one used to have to know the full stack to work in web development, now there are jobs and career paths open to us that just pertain to the client side. Ultimately, as front-end web developers, we will have a weighted interest in mastering the client-side domain. We already have a starting foundation on the front-end languages, so we want to round out our front-end knowledge by learning about how web requests are made from a client application, how the client side deals with the responses from the server side, what happens during a request, and really understanding and becoming intimate with the different browsers and the role they play to render content and execute interactions. Learning paths for all these will be discussed in the upcoming clips in this module. So areas of focus for further learning in the client-side topic include knowing server-side rendering, knowing about static site files, and knowing about bundling web apps for mobile. In the next clip, we will discuss the server side of things and identify what we need to know about that part of the landscape, including web servers and server hosts.

Server Side

Look, let's not get it twisted. Just because we can now focus on the client side and become career front-end developers, doesn't mean we can turn a blind eye to the server side. After all, we still probably need the server side to deliver our client-side code, and we definitely need to rely on it to get our data and content. But as client-side developers, we don't need to concern ourselves with how it does all that, do we? Well, no. We don't need to have a strong grasp on how it does what it does, but we do need to understand how the client uses the server side, and we should have some basic understanding of how a website is hosted and served up. Let's begin with serving it up. We know that an HTML file can be opened up in a web browser and it will render, and we have probably spun up some sort of web server on our local machine at one point in learning front-end web development. Maybe it was using NodeJS or Ruby, or maybe we use an IDE like WebStorm to launch a site. All of these are what we refer to when talking about serving up a site. They involve some sort of web-server logic that will listen on a specified port on the machine it is running on, and when requests come in on that port they will make some decisions about what website code they should run. That code may just be an HTML file that the web server will read and return to the requestor, or it may be a server-side script or code that it will know how to run, which in turn will do some work like constructing some final HTML output or data. That code will tell the web server to send back a response, usually with a specified HTML status code and some content. So these are the ways that a website and website content is served up. But what about hosting? What does that mean? Well, a web server needs to run from some machine or service, and it needs to be connected to a network that can be reached from the web. The web server is hosted on a machine, and the hosting exposes access to the web server. The standard port that web servers usually run on in a host environment is port 80 for HTTP requests and port 443 for HTTPS requests. These are known ports for web traffic on hosted machines. As a result, users don't need to include the port in the web address to access the site. So, as front-end web developers, areas of focus for further learning include familiarizing ourselves with the web servers out there and how they are hosted and run. Understanding these, even from a high-overview level, will empower us to make stronger development decisions, as well as be able to diagnose and debug issues on our own without needing to call IT. In the next clip, we will learn what we need to know about the transport part of the network, how that web address gets routed to the right hosting machine, and how the response gets delivered back.

HTTP and DNS

Okay, we've got a grasp on the client side, and we know what's going down on the server side, but what actually connects the two? We know there's a website address involved, and that allows us to request content, and when we write front-end code we are writing the content that is getting delivered as a result of that request, and we may also be writing front-end JavaScript that can make new requests from the client. But what else is going on? What else to we need to know about? A lot actually. The communication between client and server has many important elements that we must have an understanding of as front-end web developers. Let's start with the web address. Every page and resource on the web has an address that indicates where it's located. When a web browser or client makes a request for the address, it is telling an internet service provider, hey, get me content located here. The internet service provider then attempts to go down the path of retrieving content from that location by resolving that address into something it and the network can understand. It does this with DNS. DNS, the Domain Name System, is the mapping of domain names to server IP addresses. There are a couple of things we need to know about DNS as front-end devs. If we are doing contractor freelance work, we may find ourselves in situations where we are responsible for registering domain names and setting them up for sites we are building. And if we are working at a position in a company, we may be needing to wire up some client-side code to make a request to another domain. When DNS records get configured for a domain name, they get updated at one server location and then get propagated out to other internet service providers over time. Typically this doesn't take too long, maybe a day or two at most, but it is a factor that comes into play when we go to use our code that we create to make web requests to a domain name. The other thing we need to know about DNS is that there is a local file on our machines that allows us to override DNS settings from our internet connection. This means that we can create our own DNS entries that will tell our machines to go to an IP of our choosing when our client app requests a matching domain name. This is extremely helpful for front-end web development because it allows us to test our site code using full domain names without having to have our code deployed to the actual server that is hosting the code. The things we need to know about DNS as front-end devs are DNS records need to propagate, and it is possible to change local DNS so you can dev with actual URLs. So that covers addresses and getting the content, but what about the process of the request and multiple requests? What do we need to know there? The biggest thing is web requests are stateless. What is meant by stateless? When a request is made, a response is returned, and both the server and the client had no obligation to have any further communication, and each request and response stands on its own. Web requests are everywhere, not just the initial page request. A link to a style sheet or JavaScript file, an image source attribute, a font, these are all web requests. Now some of these get cached by browsers after their first response, but they still result in a request at least once. Each web request consists of headers and a body, which could be empty, so all of these requests have at least some payload cost to them. Understanding when, where, and how many web requests to make is an extremely important skill set for front-end web development. We need to have a solid understanding of the request and response lifecycle and the intricacies of them. So areas of focus for learning include understanding the role of DNS, how to configure local DNS, that web requests are stateless, and to really know the request and response architecture. Okay, on deck for the next clip we will go over the different languages of the web and learn that it is important that we get exposure to different ones.

Languages of the Web

As potential front-end web developers, we are familiar with HTML and CSS, as well as JavaScript. We know that we are going to have to learn these inside out and the roles they play in the landscape of the web, but there are other languages that we will come across and most likely work with at some point. These are known as server-side languages. Let's take a look at what we need to know about the client-side languages and then cover some of the server-side stuff. So HTML and CSS are used by browsers. Essentially these are instructions for the browser to follow to render some desired content and design. Most browsers render most HTML and CSS in the same fashion, but there are some variants. In the next clip, we will touch on this in more detail. HTML is the structure, the bones of the content. The CSS is the styling, the outfit for the content. The CSS can also dictate animation. JavaScript provides interactivity to the browser. It tells the browser to make changes to its rendered HTML content, its styling, and more. It is also used to request more content. JavaScript has been referred to as the language of the web. So, the three of these, HTML, CSS, and JavaScript, make up the trifecta of front-end languages. Each of these can actually be found on the server side because it's the server side that typically delivers these to the client application. While CSS and JavaScript are usually delivered as resources or served up as a request for a file directly, HTML can often be found as being delivered from some other language code, a server-side language code. For example, a website built using WordPress consists of PHP scripts that use PHP code to do some logic work on the server, like calling into a database and getting some data, and then constructs the HTML by mixing in dynamic data and conditional buildup of elements using PHP code along with static HTML that was written to output the final HTML that will be delivered in the response of a web request. This is known as server-side rendering. As a front-end web developer, we may be tasked with building the static HTML portion to begin with, but most likely we will end up learning some bits of PHP so we can mix in the dynamic data and conditional rendering that constructs our final desired output. Another server side scenario we will face is the case in which we need a host web application for our front-end code. Let's say we are building a client-side web app using a framework like Angular, Ember, Auralia, etc. A website can be built using ASP. NET MVC powered by the C# language on the server side to handle URL routing, dynamic resource delivery, and user authentication. That server-side code can then control delivery of the front-end code for the client-side web app. The server-side code provides a bunch of additional features and control of the other web stuff needed to make up a website or application as a whole. Node. js is another server-side architecture powered by JavaScript, which can be used to handle website stuff on the server side. While it is possible to have a website project that only consists of HTML, CSS, and JavaScript files, it is more likely that we will find them built upon a server-side language that mixes in the buildup and delivery of these client-side languages, and there's a very high probability that we, as front-end web developers will need to be able to at least handle some basic coding and be familiar with some of these server-side languages. As we begin our journey to front-end development, we should take a look into these server-side languages, play around a little bit, and get a general understanding of what is out there. When we land a job or client project, it will generally come with some server-side language already in use. At that point, we will be able to identify what server-side language we want to focus on learning on. But until that moment, we need to make ourselves aware of the common languages, but don't need to learn any one in depth. So languages and server-side architectures for us to check out are things like PHP, C# in. NET, Ruby, and Node. js. In our next clip, we are going to discuss browsers and reveal all the intricacies that we need to know about them.

Browsers

The web browser is a front-end web developer's canvas. It's the app that renders all of our hard work, all of our creative constructs. There are multiple browser vendors, and as web developers it is important that we know what is out there. There are the major players, Google Chrome, Microsoft Edge, Apple Safari, and Mozilla's Firefox. The Opera browser also has user traction. In addition to knowing the offerings, we need to understand that browsers have different ways of going about rendering the markup, styling, and code we want to write for it. The major players are all working towards standard support for HTML 5, CSS 3, and ECMAScript 6. But not all features of all these specs are supported in all of the browsers. As front-end web developers, we need to be on top of what is supported where so we know how to correctly build functionality and fallbacks. We need to be able to track down what HTML tags and features are supported in each browser so we can know when we need to leverage polyfills or fallbacks to provide cross-browser support. There are several sites like CanIUse. com and HTML5Test. com that we can keep in our tool belt to monitor feature support as we craft front-end projects. We need to understand what vendor prefixes are and how to use them for delivering common styling features across multiple browsers. We need to be familiar with the fact that browsers run different JavaScript engines. For example, Google Chrome runs the V8 engine, Microsoft Edge runs the Chakra engine, and Firefox is running SpiderMonkey. Each of these engines perform differently from one another when it comes to the speed of execution and things like garbage collection and memory management. There may be times when we write some JavaScript that runs fast in one browser, but brings another one to his knees. We need to learn to identify performance issues in our code, not only as it pertains to the logic we write, but also to the browsers we run it on. These JavaScript engines share a lot of the same APIs, but there are some variations in both API signatures and supported features. We should get some exposure to some of these differences so we can understand why it is important to always be cross-browser testing on our projects. It is also key that we know about the different mobile browsers and the different nuances that need to be addressed to properly support web applications and sites on those. Mobile browsers, both on smartphones and tablet devices running a mobile operating system, are locked into two resolutions for portrait and landscape mode. As we develop front-end markup and code, we need to be aware that when it is run in browsers on these devices the user will not have the option to resize their browser window. Browsers on these devices support features like zoom and touch, so we need to understand how to include markup and script that puts our front-end code in a state capable of handling these and provide a great experience. Just like a painter has a solid understanding of paper stock, cloth, and other mediums that they paint on, so too must we have a solid understanding of the browsers that we write code for. So areas of focus of learning include the different browser vendors, feature support of HTML and CSS, vendor prefixes, JavaScript engines, and mobile browsers.

Summary

In this second module, we covered the landscape of the web. We broke down the client side and the server side, compiling a list of terms and concepts that we need to know and be strong in as we begin our foray into the front-end web development. From the architecture options for the client side that include server generated to static resources to package code delivered to run on mobile devices, we found areas to focus our learning on, as well as opportunities for specialization in the field. As we stepped through the server side of things, we saw the importance of knowing how a site is hosted and the role of the web server as it relates to the front-end project code we will be working on. Since we will be launching web servers locally as we craft projects, it is imperative we know not only how to do that, but have gained some experience and exposure to the different web servers available to do that with. When we covered HTTP and DNS, we saw we need to know about the role DNS plays to resolve web addresses to network locations and how our workstations have local DNS that we can control during a development stage to test out domains. We also learned the importance of understanding what makes up a web request, and how being aware of the effects of things like number of requests and size, plays a key role in our ability to craft solid front-end code. A review of the languages of the web revealed that we need to get our hands dirty and expose ourselves to different web languages out there beyond just the trifecta of HTML, CSS, and JavaScript. As we take the plunge into front-end web development, we will most likely find ourselves working on projects that include some level of server-side rendering with a server-side language. Finally, we wrapped up the module with a discussion on browsers and the terms and topics we need to know about them. From feature support to JavaScript engines and mobile constraints, we need to build a deep understanding of the canvas for which we craft our front-end work of art. In our third module, Front End Development, we will drill into the core of front-end web development and learn of several specific topics and terms beyond the basics that we need to master in HTML, CSS, and JavaScript. We will also learn of the tools of the trade and figure out what we can do to prepare ourselves to adopt and become proficient at whatever our future job requires us to use in our daily project workflow.

Front End Development

Introduction

So we learned about the history of web development and some potential career paths in module 1, and we covered the lay of the land for web development in module 2. We have a good understanding of the skills in those areas that we need to bring to the table to start a career in front-end web development, and we can get to work on learning those. In this module Front-End Web Development, we will go over the languages and the tools that will make up our everyday workflow experience as front-end web developers. Beginning with the front-end trifecta of languages we will identify what we need to know from the terms and concepts, to syntax, and more, and figure out some learning paths we can take to master HTML, CSS, and JavaScript. Then we will dig into development tooling in web browsers and see why they will become and oft-used tool in our tool belt. From there we will transition over to the core tool in our arsenal - text editors and the integrated development environment, or IDE. We will learn the different offerings and what makes knowing and using them a key to a successful career in front-end web development. Finally, we will finish off with the topic of version control and learn how to use this very important workflow component. So let's get on to the next clip and explore what we really need to learn to ratchet up our HTML knowledge and come into the career game primed for success.

HTML

HTML is one of the 3-core arteries that fuel the heart of front-end web development. There are no two ways around it; we need to plan on becoming a master of HTML. What is it about HTML that we need to master? Is it just the syntax, the browser capability, or is there more? The answer is a bag full of yes. Yes we want to master the syntax. We need to be proficient with the terms and be able to talk about HTML. Sure, we know what elements are, and attributes, and form fields, but we need to get deeper with our knowledge on HTML elements and DOM. For example, we need to understand the difference between attributes and properties. Where attributes are used to store initial values, and properties are set up in the DOM to track the current values as they change over time. We need to master the different elements and should focus on the HTML 5 spec. We need to know the classics like divs, and spans, and lists, and even tables, and we need to understand the future; the new semantic elements in HTML 5 like article, aside, header, footer, section, and more. When we discuss the term semantic markup, we should understand that it is referring to element names and structure that clearly defines meaning and intent to the browser and to the developer. So elements like article, section, form, table, and image, are semantic, and elements like div and span are non-semantic. We may have not yet run into the concept of HTML nesting rules, but we will, so we should jump on learning about it. For example, a div tag cannot be nested within a span tag. Web components are a new feature that are a portable, reusable component of HTML, CSS, and JavaScript. They are a concept that attempts to solve managing and reusing chunks of website elements as well as isolating style rules and scripting they use from the rest of the page. Web components consist of custom elements, HTML templates, the shadow DOM, and HTML Imports. These are all terms and concepts we need to explore and learn. While web components are not completely supported by all browsers yet, they will be. Speaking of browser support, we need to get familiar with browser capability and feature support. Not all the HTML 5 spec is supported by all of the major browsers. As we build out front-end web code and look to use elements and functionality, we need to plan for feature detection and fallbacks, or polyfills, if we plan to support multiple browsers. We also need to think about mobile browsers and the features those support or don't support. When it comes to mobile browsers, it will pay off to know about responsive design and progressive enhancement. Responsive design is the concept of building web code that reacts and adjusts the content and layout on the page, based on the window viewport dimensions. For example, a website should be able to handle a switch on a mobile device from portrait to landscape mode. Detecting the new width of the viewport and adjusting the content layout appropriately for a great viewing experience on Bull. Progressive enhancement is the concept of detecting what features, like HTML elements and APIs, are supported by the client's browser and then providing an enhanced experience based on those elements and features available for the browser while still providing an accessible site for browsers that have less feature support. An example being the ability to detect whether the browser has touch support, like on a mobile device, and exposing user interaction that allows a swipe to the side of a list element to delete it instead of just a click on a Close button. Ultimately we should take pride in our craftsmanship. HTML is our final output; the piece of art we put on display. We should strive from the beginning of our career to craft clean, organized, and elegant markup, with a focus on minimalism as often as we can. Our HTML will always be delivered over the wire to the client side and the browser will be interpreting it to render content, so we need to value and respect size and speed, as they are a big part of the equation. Okay. Areas of focus for further learning include the HTML 5 spec and semantic markup, how the DOM works on initial load and over time, feature detection, responsive design, and progressive enhancement. In the next clip, we will go over the CSS terms, concepts, and utilities, that we should learn to prep ourselves to enter the field of front-end web development.

CSS

CSS is such a key component in front-end web development and really deserves attention to detail and a strong understanding from those that craft it. It is one thing to be able to get a website or application laid out and looking pretty with CSS. But fully understanding the browser rendering engine and the decisions it makes, well that's a huge part of the job of front-end web developer. We have to dive in and learn all we can about CSS. Just like HTML, CSS is a first-class citizen for a front-end web developer. So let's cover the terms and concepts we want to make sure we know when it comes to CSS. Things like, specificity, inheritance, specified, computed, and actual values, the box model, and Flexbox, as well as other things in the CSS world, like style guides and preprocessors. We will begin with terms, starting with specificity. Specificity is a weight that the browser applies to a CSS declaration based on the count of each selector type. From that, the browser decides which CSS to apply, and in the case of a tie, which rules when. To enter the field being strong on CSS, we must spend the time to learn all about specificity and the logic that governs it, so we can fully understand how the browser is applying CSS, and how to write CSS effectively to accomplish what we intend to build visually. Inheritance is another term that we need to be very familiar with when it comes to CSS. Some CSS properties are inherited by default, and some are not. When properties are inherited, the child HTML element will use the closest parent element style up the tree when the child element doesn't have that style rule. For example, the color and font rules are inherited. The last element down the tree that has the color style rule will dictate the color for all its children down. The border rule is not inherited and will only get applied to the targeted element. This is inheritance. Something we need to know and understand very well to become proficient in front-end web development right out of the gate. We need to know the difference between specified, computed, and actual values, in CSS; where the specified being the value from the rule, inherited are default. The computed being a value based off of a relative value, like EM units or percentages, and the actual being the final value used after any computations. EMs and percentages are another part of CSS that we need to learn, especially how and when it makes sense to use one or the other. We should be very familiar with the box model in CSS. The browser's rendering engine works with each markup element as a rectangular box and attempts to determine size and properties based on the CSS we write. We need to understand how margins, padding, and borders, on elements affect their visual display and those elements around them. Flexbox, or flexible box in CSS 3, is a layout mode designed to make it easier to have control over how elements are arranged on a page based on different viewport sizes. Using Flexbox is a key component to doing responsive design and building websites that work well across devices with different viewport dimensions or changing dimensions. As front-end web developers, we will constantly be battling layout challenges and we need to be armed with all the knowledge and tools we can to meet those needs head on. So knowing the terms and understanding the rendering engine will empower us to come into the industry strong. We also need to come in organized. One of the ways we can achieve this is through style guides. Instead of just firing from the hip and coming up with our own, we should spend the time to research different style guide concepts that have been discussed and adopted throughout the industry, concepts like SMACSS and OOCSS. SMACSS, or scalable and modular architecture for CSS, is a style guide for tackling and maintaining CSS in a modular way to support a website or application that will have a large growth in CSS over time. OOCSS, or object-oriented CSS, is an approach to writing CSS with a focus on reusable chunks and using a composition of those chunks to style elements. These style guides help to lay a foundation for decisions such as, naming conventions for selectors, and structure of CSS. All common challenges that front-end web developers have faced since the dawn of the web. While one of the beauties of the web development is the ability to shape it as we sit fit, it is advantageous to explore, learn from, and even adopt patterns and practices of those that have come before us, as they have been doing battle with CSS and have solved many of the problems and challenges that we will face as we come into the industry. The final thing we need to learn up on is preprocessors. SASS, LESS, and Stylus to name a few. A preprocessor attempts to allow a developer to write CSS in a different fashion that is easier to organize and maintain, and the preprocessor will handle converting that into valid CSS. There is a good chance that the company we start out career at will have some sort of CSS preprocessor incorporated into their projects, so we should check some out and understand how they work and what that development experience is like. And, like we emphasized in our previous clip on HTML, above all we should take pride in our craftsmanship of the CSS that we write. From the beginning we should focus on size and elegance and learn to write efficient CSS that has a small footprint. CSS is another piece of content that gets delivered over the wire to the client side and gets processed by the browser, so size and speed absolutely matter. Okay. Areas of focus for further learning on CSS include knowing all the terminology, style guides, and preprocessors. Coming up in the next clip, we will dig deep into the terms and concepts in JavaScript that we probably didn't learn at a boot camp or initial front-end training, but are absolutely key to everyday JavaScript development.

JavaScript

JavaScript is such a big part of the equation today, that it is really expected that front-end web developers are proficient in it. To not know JavaScript, or to only know enough to get by, would put us at a huge disadvantage. In fact, many front-end job listings are starting to use the term front-end engineer instead of front-end web developer. So what is it that we need to make sure we know about JavaScript, and what can we do to make sure those skills are sharp? When we go to interview for a position, we are going to get asked about our level of skill in JavaScript. We will most likely need to read and analyze some code. And when we land a job and begin working on project code, we will be expected to be able to write clean, performant JavaScript that is mindful of things like delivery size, memory footprint, and DOM manipulation impact. Let's go over the elements of JavaScript that we will want to make sure we understand thoroughly when we enter a career in front-end web development. We will also focus on some key JavaScript terms we should be familiar with as we go over these elements. To begin with, we should already be familiar with the fact that JavaScript is a dynamic, or loosely-typed language, meaning a variable can be set to one data type and later changed to another. And we should also be familiar with writing and reading JavaScript code, creating functions, DOM manipulation, and event delegates. We've most likely learned all these during a boot camp or course on JavaScript, but to really know JavaScript we have to be familiar with more than just that. There are five main concepts in JavaScript that we are going to need to know very well. Some of these we may have done in practice before, but not have learned the proper name or terminology. It will be important that we learn those prior to starting our career. So let's go over the five. They are data types, prototyping, scope, closures, and the this keyword. We'll begin with datatypes. As of ECMAScript 6, there are six primitive datatypes in JavaScript. These are string, number, Boolean, symbol, null, and undefined. Primitive datatypes in JavaScript are immutable, meaning that once they are set they cannot be changed. The other JavaScript datatype is object. An object in JavaScript is a collection of properties. Setting a variable to an object using the curly brace notation, as in this example, is called an object literal. JavaScript has some standard built-in objects and in fact, every data type is derived of an object type. Which brings us to our next key concept, Prototyping. An object type can be defined by creating a function statement. For example, we can create a user-object definition by creating a function named user and setting some default values on it. Then, using the prototype property which all objects have, we can add functionality by adding a new property like getFullName and setting it to an inline function. Prototyping in JavaScript allows us to write more performant code. When creating a new user, the memory footprint for an instance of a user only needs to contain the first and lastName properties. The getFullName is on the prototype property which the JavaScript engine will keep track of separately from the object instance. The third concept we need to understand, and one of the most important, is scope. We are familiar with the concept of scope, but maybe haven't heard it called by its name. When we enter the field of front-end web development we will hear the term all the time, most likely as early as interview questions, so we need to get comfortable with scope. In JavaScript, scope refers to the variables and functions a particular block of executing code has access to. For example, when a variable is declared within a function, it is scoped within that function and is not accessible outside of that function. We need to be well-versed in the rules of scope and the scope chain. Which leads us to the fourth concept, closures. A closure is the concept that an inner function has access to its own scope, the outer scopes function variables and functions, and global variables and functions. Those are the three scope chains of a closure. Closures are the powerhouse of JavaScript development. As front-end web developers, we will need to know them well as they help to isolate code and avoid collision, especially as we find ourself leveraging third-party libraries and code, mixed in with our own handcrafted code. Finally, we need to get comfortable with how the this keyword works in JavaScript. If we are well versed in scope and closures, it is easy to master the this keyword. We need to be aware that in a function, most of the time this is based on the object that invokes the function. It is a shortcut reference to the invoking object. Determining the invoking object is the tricky part. Nevertheless, we must take the time to learn all about the this keyword in JavaScript to be successful in a career in front-end web development. And, just like in HTML and CSS, we want to form pride and craftsmanship when architecting and writing our JavaScript. Performance, speed, maintainability - they are all things we will achieve by having attention to detail and passion for what we build from the very beginning. Alright. So areas of focus for being well-versed in JavaScript include datatypes, prototyping, scope, closures, and the this keyword. In our next clip we will cover browser developer tools, and what we need to know beyond the ability to inspect elements and see JavaScript errors.

Browser Tools

Browsers allow us to do a lot of investigation and prototyping within them via the developer tools. We need to become familiar with them as they are a part of the core tool set of a front-end developer and used pretty much on a daily basis. We are probably familiar with the developer's tool panel and have most likely used it for element inspection, from the elements tab in Chrome, the inspector tab in Firefox, or the DOM explorer tab in Microsoft Edge. And we have most likely used the styles or rules tool to test and tweak CSS rules on the fly on a site. Maybe we even hit up the console to see JavaScript output of errors. But there is so much more that we must learn to utilize when it comes to browser dev tools. We need to become familiar with the Network tab so we can identify requests that are being made by our web apps, and see their impact on load times and data size, and use the Headers and Response tabs to get visibility on the structure of requests from our apps, and what the server side is returning to us. The Sources tool on the top level menu in Chrome and under the Debugger in Firefox and Edge, provides us the ability to look at JavaScript files and inline scripts that are a part of a web page. We can view the different scripts and add breakpoints, and then reload the page and debug JavaScript in the browser with full support for viewing the call stack and checking scope variable values. We can even jump to the Console during a break point and execute JavaScript. Speaking of the Console, we can use this to execute JavaScript at any time, not just during debug breakpoints. We will find ourselves spending a decent amount of time testing out script code, or even running DOM selector code from the Console to track down elements or initiate events. Cookies and local storage can be inspected from the Resources tab in Chrome, and the Storage tab in Firefox, which needs to be enabled from the Options menu first. When we write code to set cookies or modify local storage, we will use these tools to check our data and make sure it is changing as expected. In addition to these, there are other tools to monitor in profile performance. We may not touch these early on in our career, but it won't be long before we have the experience and understanding of JavaScript that will empower us to really understand how to read and leverage the performance tools to craft top-notch scripting. So areas of focus of learning when it comes to browser dev tools, include the Elements, Inspector, DOM elements tab, the Network tab, the Sources/Debugger tab, the Console, and the Resources/Storage tab. In our next clip we will go over our primary tool in our front-end web development tool set, the text editor, an integrated development environment, or IDE.

Development Environments

Text editors and integrated development environments, or IDEs, are the tool of choice for accomplishing the majority of front-end web development tasks. Sure, one could use a simple text editor and build front-end solutions just as good, but most web shops that employ front-end developers will have a chosen IDE, or set of IDEs that they want their employees to use. So let's start with text editors. Text editors like Sublime, Atom, and Brackets, are rich tools for writing code files with support for syntax highlighting, auto complete, and task runners. We probably have used one of these as we learned HTML, CSS, and JavaScript. They provide a great way to get into code quickly and get right to creating, and while it is likely we will be using text editors occasionally in our tool belt in our career in front-end web development, most of our project work will probably center around an IDE. So what is an IDE, and why do we need to know about them as front-end developers? Let's start with the what. IDEs for web development, like WebStorm and Visual Studio, provide all the same features of a text editor: syntax, highlighting, auto-complete, etc., but they also have project-level knowledge of files and code within those files. So IDEs can provide rich intelliSense, or context aware code completion and code refactoring. Let's go over those, because they are two terms that are used almost daily in the industry, and those two terms we really need to know. IntelliSense is similar to auto-complete, but it is based on the application actually understanding what items we are working on and providing us with valid options to use. For example, if we are in JavaScript and we are trying to access a property on a variable, when we type a period, the intelliSense will show us only properties that are actually available for that variable. Or, if we were inside of an HTML image tag and started to type the letter S, we would get intelliSense showing the option for the SRC attribute, where in a div tag we would not. Refactoring refers to the action of changing code, usually in an attempt to improve or adjust functionality. In an IDE, which has project-level knowledge of code, there are refactor tools that will handle tracking down usages of the code we change and will handle making sure it is changed in all the right places. For example, changing a CSS selector name in a file in a project, using the WebStorm IDE, will result in WebStorm tracking down that selector and other files and changing it there for us all in one shot. Refactored tools are one of the big differences between IDEs and text editors, which typically don't have cross-file refactoring support. Okay. Why else do we as front-end web developers, need to know of and be able to use these IDEs? One of the biggest reasons centers around collaborative workflow. If you find yourself working on a project with multiple people, the need to work in a common pattern and share common resources becomes of high importance. Having different IDEs can result in different project settings and structures. Most team projects in the web development world center around an IDE. Now, there are certainly exceptions, as well as ways to make multiple IDEs play nice in a single project, but the chances of having the option to explore those dwindle when you find yourself working for a company, or even doing contract work for a company that has an established development workflow. Our key to success when it comes to text editors and IDEs is to jump on being aware of what is out there and getting our feet wet on a regular basis. That underlying theme of the web development industry that change is always around the next corner is prevalent in the text editor and IDE landscape as well. As we travel from project to project, our chance of needing to use a different code editor is high. Whether we are changing companies or our company is taking on a new client job, or even if we were doing contract work and landing new bids, we can easily find ourselves needing to use a different text editor or IDE. Sometimes this centers on a language or maybe around a source-control plugin, or even based on if a company is a Microsoft shop or an Apple shop. Most of these code editors provide similar functionality, they just differ on things like shortcut keys and user interface. Learning one makes it pretty easy to jump into others, so we don't need to be concerned about picking them all up as we start our career, but we should find excitement and passion in wanting to check out others and get in there and play. After all, these are the main tool of our trade. Let's wrap up this clip on development environments by going over the terms we need to be familiar with. There is text editor and IDE, with text editor being the program to write code files, and an IDE being a program to work on code projects. Then there's intelliSense, which is context-aware code completion, and finally, refactoring - going back and changing code with renames and structure alterations. So areas of focus for further learning include getting exposure to different text editors and IDEs, and working with full projects in text editors and IDEs and learning the differences between the two. In our next clip we will cover what we need to know when it comes to version control, a way of recording changes to files in a project over time.

Version Control

Version control is like backup for our code. It also tells a story of the journey we go through when crafting a web project. Sure, we can work on project files located on our machine without a version control solution, but we would find ourselves way out of touch with the current state of the web development industry. To take on front-end web development as a career, we must learn how to use version control to manage our projects. Companies use it, individuals use it; we need to learn to use it. But what exactly is version control? Version control is an application or service tasked with storing copies of digital content, whether it be code files, resources, etc. Sets of digital assets, like a website project, are stored in a container called a repository. To send files to a repository we do what is called a checkin where we check in a copy of the files. Each time we check in we have a chance to add notes or comments about the file changes we made. Before we start to work on some project changes, we do a getLatest from the project repository. This lets us get the most current code base for our repository, ensuring that we have received everyone else's latest updates to the code. When we are ready to work on a file or asset, we do a checkout which indicates we are modifying a digital resource. There are several version-control options out there, like Git, TFS, Subversion, and more. We need to learn up on at least one of them and get exposure to the workflow and concepts. Git is a good one because it is used by GitHub and Visual Studio Online, services that offer free repositories and are easy to get up and running with. There's a lot to version control and it can feel a bit daunting to take on, but it is commonplace in the industry to be using it, and we will be expected to be able to use it from the beginning of our front-end web development career. So areas of focus for further learning about version control include, learn the terminology and try out a version control system like Git.

Summary

In this third module we covered the finer details of front-end web development and all that we really need to know to get jump-started in a career. We began with a discussion on HTML and saw that there's a lot more to it than just markup syntax and knowing how to use elements. We formed a solid list of topics and terms that we can focus on learning to put our HTML knowledge on a strong level as we enter the field. Then we talk about CSS topics and terms that we need to be on top of for our career in front-end web development. From selectors to supported features, and concepts like naming conventions and preprocessors, we compounded a list of topics to further our learning on CSS and prepare us to work with it on a professional level. We rounded out our focus on the trifecta of front-end languages by really understanding all that we should know when it comes to JavaScript. We talked about knowing datatypes, prototyping, and how scope works. We capped this module off by looking at our front-end web development toolset, from browser dev tools to text editors and IDEs, to work with markup and code, to version control to maintain changes of our code over time. With that, we've covered all the terms, concepts, and topics, that we need to prepare for as it pertains to the daily workflow in the life of a career of a front-end web developer. In the next module, Getting in the Door, we will cover how to prepare for and succeed in landing a job at a company doing front-end web development.

Getting in the Door

Introduction

We have covered the history of web development, figured out what we need to learn when it comes to the landscape of the web, and have gone into detail as to what is needed to master it when it comes to the domain of front-end web development and the tools of the trade. So there are two things left to do: Go over the keys to land some work, and figure out a plan for how we can grow our career once we do. This module, Getting in the Door, will cover how we will go about landing a position in the front-end web development industry. We will cover what we need to be thinking about and what prep work we need to do prior to going out and interviewing for a position.

Company Career Path - Prep

We want to learn about the interview process for landing a job in the front-end web development industry. But before we talk about that, we need to cover things we can do to prep for interviews. There are two phases outside of a resume or CV to getting prepped for interviews in the front-end web development industry. The first involves exploratory information an employer will look for during the vetting process when they are doing a background check on us based on the up-front info we provided them. The second involves making sure our knowledge on the web development landscape, outside of the core front-end languages of HTML, CSS, and JavaScript, is fresh in our minds and that we are prepared to have a conversation around that landscape. We won't go into detail in this course on the building of our resume. We can search Pluralsight for the term resume and find some great courses on the topic to watch. Our resume, or CV, is only one part of the equation that an employer factors into the hiring process and it is usually only looked at in the beginning to decide whether to bring us in for an interview or not. Most web development companies are savvy enough to know that the listing-out of languages and frameworks on our resume doesn't really mean anything. A discussion over those and some same code will be more telling. What else will a potential employer be looking at of ours before our interviews? Sample projects we have worked on and made available, for one. Community involvement we have participated in, like user groups we attend, social media posts we make that involve web development content, and if we are active in sites like Stack Overflow, looking to ask or answer questions on front-end web development topics. So one of the most valuable things we can do is get a sample project up and going prior to the interview process. We can create a web project and use GitHub to host our source code for it. We should look to use either and IDE, like WebStorm or Visual Studio Community Edition, or a text editor like Sublime Text or Visual Studio Code, that has some Git support or plugin for checking in code to GitHub. The project doesn't have to be anything crazy advanced. The purpose of it is not to showcase our HTML, CSS, and JavaScript skills, but rather to show we understand the overall workflow that will be involved in day-to-day operations of a front-end web developer. If we get the opportunity to seek out and attend some local user groups prior to hunting down employment, we should list those in either a cover letter or an email that we send with our resume to an employer. Attending these shows an employer that we are investing in self-education and growth. It also gives them the additional piece to look up prior to our interview. The more elements that an employer has to investigate prior to our interview, the richer the story they will build around us which we definitely want, as this will help us stand out, as well as give them more topics they can discuss with us during the interview process. If we have setup a Twitter account we should get on it and tweet out front-end web development content. Start with retweeting content that is relevant and useful. We should also discover content on our own, maybe blog posts or a good podcast and tweet about them.

Again, this gives potential employers data to investigate and helps them build the picture of what we are all about. If we are actively tweeting and retweeting content on say, responsive web design, there's a solid chance that the topic will come up in the interview providing us with an opportunity to illustrate our passion and knowledge. So by being active prior we can actually control some elements of how that story is built of us. Stack overflow is a free-to-use question and answer website that is used almost on a daily occurrence by web developers. People post programming related questions and the community posts answers. Anyone can ask a question and anyone can help out with an answer. This site is not only a great place to learn more about front-end web development and how to solve specific issues that arise, but it is also a great place to contribute to the industry and to showcase our skillset. Spending some time to post some questions and answers that we feel knowledgeable about can be a big benefit to our career opportunity. When we have answers that we have posted, we have provided employers with the ability to see and learn about our skills at a micro-level. How we are able to solve specific problems and what our code looks like when we do so. For an employer to be able to learn that about a potential employee prior to getting them in and working, is a big plus. Landscape. We need to be prepared to recognize the lingo and talk about those topics. We should feel pretty confident about our ability to talk on the trifecta of front-end languages since we spent the most time honing those skills, and the interview process will most certainly putting those to the test. But companies need front-end developers to come in knowing more than just that, so they will be plenty of questions and discussion around the web landscape as a whole to uncover our range of understanding. Several of these questions will be phrased around terminology and vocabulary used in the industry, so we need to be able to identify those terms and understand what they refer to in order to stand out. The things we covered in this course prior to this module addressed those. Employers won't be expecting us to be experts on these topics, and when discussing them we should not hesitate to make it clear as to which we know well, and which we are aware of, but would like to learn more on. But for sure, we need to be able to follow along. So when we build out our resume and reach out to potential employers, we need to make sure we list all the prep work we do. Include links to our projects on GitHub or better yet, our GitHub user account. Let them know what user groups we have been attending, include our social media handles and links, and include our link to our Stack Overflow user account. Alright. With the prep work covered, we can move onto our next clip and talk about the actual interview process.

Company Career Path - Interview

So what can we expect when we go to interview for a front-end web development position? We will have an opportunity to talk about projects we have worked on, so we will want to be prepared to do so. These will give us a chance to showcase our awareness and understanding of how a website project is set up. We will want to show experience using version control and working with front-end code that is within a full project. While not a must for gaining employment doing front-end web development, having experience with these greatly increase our shot at getting hired. One of the elements that an employer will weigh when accessing the potential hire is the amount they will need to invest to on-board someone. The training of concepts like version control and project workflow are time consuming. When we can show that we have experience with these during our interview, it gives companies and identifiable and measureable difference between us and other candidates. Another common topic in the interview process will be the discussion about text editors and IDEs. We may get asked what our favorite editor is, or if we have experience with this one or that, usually in relation to the ones that the potential employer uses in-house. If we were able to discover what the employer uses ahead of time, find the opportunity to show that we have some familiarity with it. An employer will be looking for developers who have a passion for learning. This industry is built on self-education. The employers have been down that road. We will be asked discovery questions that are trying to extract that interest out of us. We need to take every opportunity they give us to express our desire to learn and excitement over the industry as a whole and we need to avoid any negativity towards topics, languages, or frameworks, even if the interviewer takes a shot at a tech, we need to stay out of that dark side. This is an industry of constant change and flux. Employers will be looking for candidates who are willing and able to change at a rapid pace, and willing to embrace that change. We will get asked questions related to our ability to change, and ones that will attempt to draw out our feelings on change. If we want to be in this industry we must embrace and welcome change, so let's show them that during the interview process when given the chance. We will most likely be asked about what types of front-end frameworks we have experience with, or have worked on. We need to show that we understand what a framework is and why we should use one. So when asked what framework we have enjoyed working with recently, an appropriate response would be something along the lines of, I recently included Bootstrap into my project to leverage some standard UI markup and styling easier, and I also ended up using a JavaScript library called Lodash that I found useful for handling common array functions. Finally, we will definitely be asked to show our skills in the front-end trifecta of languages - HTML, CSS, and JavaScript. But these are where we have spent our mastery points and perfected our game, so we should be prepared to nail those, and we will, just like all the other candidates. That will be the easy part. Now let's recap the key aspects of the interview that we will nail that will make us standout. We need to be ready to discuss a project or projects that we have worked on. The potential employer will most likely have looked over it ahead of time. Show them that we are familiar with a project workflow. Show we know IDEs and text editors. What have we used and what are others out there? If we know what the employer uses, show we have some familiarity with it. Show enthusiasm for the industry and technology. A passion to always be learning and the flexibility to change. With these in mind, we can nail that interview and start our career.

Summary

In this fourth module, we covered what we can do to prepare for an interview for a front-end web development position. We covered working on some side projects ahead of time and getting involved with the community, like attending a user group or being active on Twitter around web development topics. We talked about the need to brush up on our knowledge of the landscape of the web and be prepared to talk about it with confidence and understanding. All of these will provide content for a potential employer to view and research before they invite us in for an interview. Employers are looking for candidates with involvement and a willingness to learn. They want to uncover discussion points that they can have when they meet with them. Then we covered the interview process and what to expect when applying for a position doing front-end web development at the start of our career. We learned that employees will be weighing in on how much it will cost them to on-board us, so we should take every opportunity that we can to show them that we are already familiar with several of their daily workflows and tools, like IDEs they use and working with version control. We also learned that it is important to show a passion for the industry and an overwhelming desire to learn and adapt, as change is the most common theme in web development. In the upcoming final module, Growing Your Career, we will wrap things up with a discussion on the things we can do to always keep learning an adapting and give ourselves a plan for success as we launch into this great new career.

Growing Your Career

Introduction

So we found a way to start a career in front-end web development. We landed that job and we are on our way. What's next? Is it all just about piling on experience as we go, and attacking each project as it comes our way? Or should we be looking to do more? In this module, Growing Your

Career, we will learn about all the different ways we can keep on top of our game. The front-end web development industry, as well as the web industry as a whole, evolves at extremely rapid pace. If we were going to have a career in this industry, we too must be able to evolve and change at that same pace. Unlike other industries that have established practices and structure and rules over time that stand as the foundation of those industries, web development is fluid and adaptive. The establishments within those other industries allows for more training material, books, and lectures, which are concrete and have been around for years, making learning and skill-building paths abundant and easily acquired. With web development always on the fringe, we must seek out information in order to learn and master our trade. That means self-educating and discovering and constructing our own learning paths on a regular basis from before we enter the industry, to every day that we are in it. Yes, there will be many sources at our disposal to learn finite things about specific components, like Pluralsight courses, and podcasts, and conferences, and blog posts, and there are some certificate programs for certain tech. But we are not going to find any big-picture story, or go to some extended education program that will teach us the law book on web development over the course for four years. The rules and landscape just change too rapidly. Even the educators and teaching material have to adapt on a month-to-month cadence. So while we are diving into our new career position and working on projects at hand, we need to be self-educating and seeking out content in our industry and the tech that powers it on a daily basis. We can't sit back and wait on baited breath for our employer to send us to some career advancement training program; that's not the way this industry works. We need to be the ones to grow and cultivate our career and skills. In this final module, we will go over all of the ways that we can do that. From written, audio, and video material to consume, to networking with other devs in the vast web community, keeping our fingers on the pulse of the industry, and always getting our hands and fingers on the new frameworks and tinkering with different approaches to code and markup. So let's roll into the next clip and get started discussing how we can keep our skills sharp through constant self-education.

Self Education

Self-education starts with a desire to constantly be learning. Where can we find content to satiate that desire as we immerse ourselves in a career in front-end web development? There are five primary sources we can hit on a regular basis to keep our skills sharp and up our game: Blog posts, podcasts, books, training videos, and conferences. Let's dissect each one and see if we can build up a game plan for putting them into our daily rotation. Blog posts. Blog posts are one of the best sources for current information, techniques, and examples in our industry. Many web developers contribute free content to the web through personal blogs, where they discuss tech in the industry and create guides for how to accomplish a task or requirement in code and design. Most of the time this comes from having just worked through some challenge that they faced on a project in which there was no guide and they found themselves needing to figure things out on their own. Creating a blog post about the problem and solution is not only a way to document and store the procedure for later reference, but it is also a way to contribute and help out others in the industry who may come across a similar problem or challenge. We will find ourselves reading blog posts as we do things like search the web for answers to problems we are trying to solve, or features we are looking to build. We should seek out content through this channel, as it is one of the fastest ways for new information to get distributed out in our industry. We also need to look to find sites that curate blog post links on a daily fashion, sites that list out the latest blog post relating to HTML, and CSS, and JavaScript, and JavaScript frameworks. We will want to develop a bookmark strategy so we can collect links to posts we come across that we will either want to read later or want to revisit. Sites like Pocket or Delicious are free to use and provide us with a way to collect links to web content. We can add tags to each saved link to help us organize and rediscover content at a later date. Podcasts. Subscribing and listening to podcasts is another great educational source for our industry. Just like blogposts, there are many web developers that create podcast episodes on a regular basis. Podcasts in our industry typically consist of episodes centered around a piece of technology or framework, or maybe an established technique, or even a new approach, with a host or group of hosts driving the discussion and usually some guest or guests who have experience on a topic of discussion. Listening to web development podcasts is a great way to discover new things in our industry, or pick up some additional details or tips and tricks on a topic we are already familiar with. They are also a great way to listen to the discussion going on in the industry and to be introduced to developers that are active in the dev community. Many podcast hosts and guests also have blogs and present at conferences or user groups as well. Books. Books are another source of educational material for us, albeit one we must have a clear understanding of how to approach. Books on web development have a short shelf life. When released they can be an excellent source of deep dive into a particular topic or technology, but most of the technology in our industry is rapidly changing, and while frameworks may live on in name, version changes at a high cadence. So a book on say, Angular 1. x, is highly useful up to the point at which Angular 2. 0 is released. Yes, the older version content can still be of use, especially if we are still working on or maintaining a site or code base that has not been upgraded to the latest version, but we need to keep this in mind as we approach books as a source of learning for our career in front-end web development. Training Videos. We are already familiar with training videos as a source of self-education. I mean, we are into the last module in this

Pluralsight course. Pluralsight videos are a great way to learn specific topics and the structure of small clips in each course allows us to jump right into and consume video segments that are relative to what we need at a given moment. The consumption is easy and effective. YouTube is another great source for training clips on front-end web development content. There are devs in our industry that post video how-to clips just as regular as others create new blog posts. Training videos are a great way to see the code creation and refactor process, and discover techniques and approaches to the development process that others use. We will not only learn a topic, but we will also catch things like shortcuts that people use in IDEs or text editors, and tools that they use to accomplish common tasks. These are elements we don't get out of a blog post or podcast, so training videos should be in our regular rotation of self-educating sources. The final source is conferences. Industry conferences are typically 2-4 days long and are centered on a particular framework or language. Sometimes these have a workshop day on the first day of the conference in which you would work through coding some examples. Attending conferences gives us a chance to step away from the day-to-day and spend a period of time focused on the tech covered by the conference. Speakers at conferences are members of the dev community sharing their experience and techniques. We get a change to network with others in the industry and immerse ourselves in the industry. But all conferences aren't necessarily at a physical location. There are more and more virtual conferences starting up where we can watch sessions streaming online, even current conferences are starting to stream content or provide session videos online a few days after the conference concludes. So all in all, there are a plethora of resources and channels for us to always be working on our personal growth and quests for gaining knowledge in the front-end web development industry. In the next clip we will learn how networking plays an important role in providing the opportunity to talk about our craft and experience other approaches to the way we build things.

Networking

One of the great aspects of the web development industry is the community around it and the constant engagement opportunities, not to mention the willingness of its members to share, educate, and mentor. A lot of this comes from the pure enjoyment at the opportunity to talk shop with others that speak the same language. While the use of the web and websites is becoming commonplace, the discussion around what it is built on and what powers it, remains something that drives people to zone out when we get into the details about it. So we crave the opportunity to engage in discussion with others that speak the same language. And by engaging in those discussions, we get the chance to hear other stories - other approaches to problems and implementations. Networking with others in our industry is an important component of our continued growth as front-end web developers. Three solid avenues for networking are user groups, hackathons, and code camps. User groups are free-to-attend local groups that meet once a month or so to discuss a particular topic. Like a single session at a conference, a topic is usually presented on by a guest speaker, or a member of the group, and discussion ensues or even takes place during the session. User groups are a great way to connect with other developers in our area. They provide the opportunity to talk shop and learn on a regular basis. They can also be a great source for contract work and making connections for employment. Hackathons are usually a one-day event in which developers get together, sometimes in predetermined teams and sometimes teams put together on the day of the event, and take on a coding challenge. There is a theme for the challenge and sometimes a required tech or library, and developers spend the day coming up with an application concept and attempt to get as far as possible on it. Hackathons are a way to network and write code at the same time. They provide an opportunity to experience pair programming, working through pressure, and delivering a project as a team. Code camps are like a free conference. Typically held over a weekend to make it more accessible to working developers, code camps are put on by other developers in the industry looking to give back. The speakers at code camps are looking to do the same. They volunteer their time to put on a session or sessions, usually around topics that they are passionate about, or that they have experience with from their daily job. Code camps are a great place to converse with other developers. The production level of a code camp is on a much smaller scale than a conference, so the speakers and organizers are more accessible. When we network with other developers, we get a chance to query content from another human being that can interpret concepts we are attempting to describe, something a search engine can't do. Plus we get the chance to share excitement and passion for our craft and get energized to do more of it. In the next clip we will discuss some ways we can stay on top of the ever-changing landscape of the front-end web development industry.

Finger on the Pulse

So we've discussed how we can get our learn on on a daily basis through blogs and podcasts and videos, and we covered how we can network with others and engage in the community. But how do we keep on top of the what? What do we need to be learning? What's hot? What is showing success and gaining traction? In this landscape of constant change, it is important that we keep our finger of the pulse of the industry, so how do we do that? Well, we can start with social media. If we don't have a Twitter account, we need to get one. Even if we don't want to be vocal on it, we should plan on watching it. So many of the leaders and industry shapers are active on Twitter; so many devs in general are active on it, sharing links to content, discussing and sharing opinions, and experiences on topics. An effective way to use Twitter is to use TweetDeck to monitor multiple streams of content. TweetDeck lets us set up columns based on different filters. If we start discovering and following other devs, we can do things like add them to lists and then create columns for those lists. So we could group devs we follow by HTML, CSS, and maybe JavaScript, to make it easier to monitor content based on domain. We can also set up some columns filtered by hashtags. We can be watching content on say, AngularJS in a column, CSS 3 in another, and web animation in another. Twitter is a great source of content for our industry. As we grow our career, we can start contributing to that stream of content. There are a handful of websites that report on industry trends and announcements, as well as aggregate recent posts around topics. If we monitor these on a daily basis, they can keep us abreast on the latest news and illuminate us on things we may want to be looking into. Up next, we will cover the importance of constantly building and trying out new concepts, frameworks, and languages.

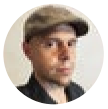
Tinkering

The final thing we can do to build and grow our career, and one of the more important, is to always be tinkering and trying out new code and techniques; whether that's trying out new frameworks in sample projects, or playing around with a run at a small feature or piece of code in a web-based code snippet application. When we read a blog post or listen to a podcast, we can open up a text editor and put together some sample code and follow along. When we watch a video course, like one of the Pluralsight courses, we can download exercise files, and explore the code, and try changes and tweaks. When we come up with an idea or thought, we can try it out without even needing a text editor or IDE, or even need to set up a project around it. JSFiddle is a web-based application that allows us to write JavaScript and test it out alongside some HTML and CSS, without the need to set up a web server to host it. Plunker allows us to use a web-based editor to build some HTML, CSS, and JavaScript, and see the output run. We can save and share those with others, and we can clone others and make our own changes to them. CodePen is a similar site that lets us work with the trifecta of client-side languages, but it also has a community share element to it where we can discover and view other user-created samples. Like Plunker, we can clone and modify those to try out our own tweaks to it. Open Source is another great place to tinker with code. There is a vast array of projects in the web development space on GitHub, from helper libraries to full projects to well-known front-end frameworks and libraries. Getting involved in an existing project is a great way to have some structure and goal when tinkering and trying out new coding techniques. Creating an open source project is another great way to push our skills forward. Maybe we take one of our sample projects we put up there, prior to looking to get hired, and we expand upon that. Or maybe we come across another idea, or find some common code that we are consistently writing in our work projects that would make a great case for a reusable library. Starting an open source project actually allows us to gain a bunch of skills early on in our career. From organizing and setting up a web project to documenting it and crafting usage examples, planning features and solving bugs, working with other devs and having community engagement, all skills that are sometimes hard to come by when working on a specific task at a company in our day-to-day work, but are there for the taking if we had the passion for personal growth.

Summary

Wow, what a journey. We've covered a ton of stuff, from the history of the web development industry, a discussion about the landscape of the web, a finite exploration of the elements of front-end web development that we need to master, how to prep for an interview for a job, and finished with a great plan for personal growth. A career in front-end web development involves so much more than just a knowledge of the front-end trifecta of HTML, CSS, and JavaScript. We saw that, and now we have a ton of guidance as to what all that other stuff is and where we can focus our next round of learning to really prepare ourselves to go out and pursue that career and make it happen. It was a real pleasure to be able to share all this insight and advice with you throughout this course on a topic that means so much to me, in an industry that has provided me with so many great opportunities for more than a decade. I'm excited for each and every one of you working hard to be a part of its future. Now go out there and get your learn on and kick start your career, and please, share your journey with me on Twitter @schwarty.

Course author



Justin Schwartzenberger

Justin Schwartzenberger is a Lead Engineer at SoCreate, currently working on the future of screenplay writing software. Justin cut his teeth on C and C++ before getting into the web development...

Course info

Level	Beginner
Rating	★★★★☆ (219)
My rating	★★★★★
Duration	1h 48m
Released	16 Jun 2015

Share course