Learning Opportunities and Challenges for Web Developers

by Devmountain Tutorials

WHAT'S COVERED

In this lesson, you will learn how to list some challenges for a web developer. Specifically, this lesson will cover:

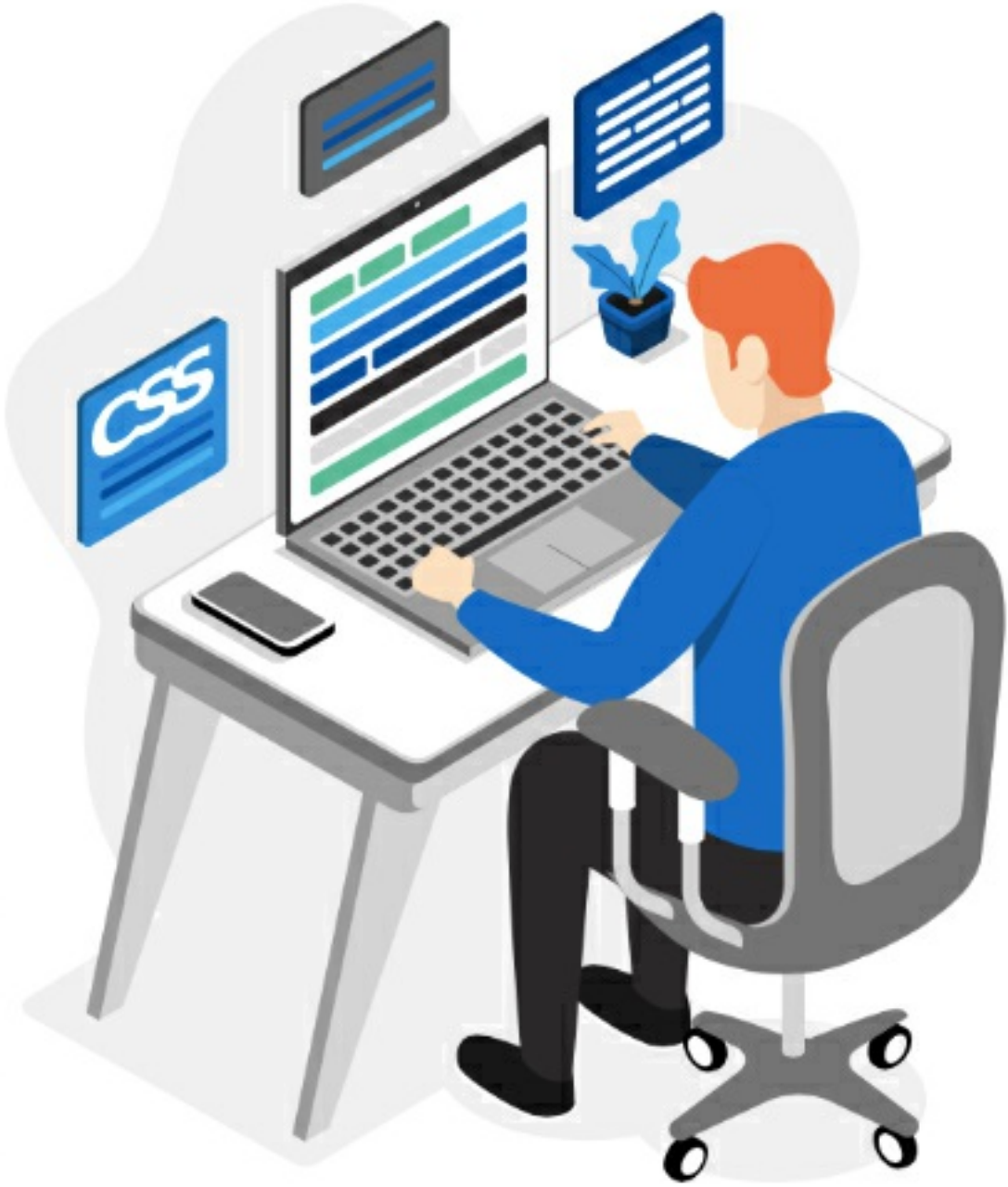
1. Website Architecture and Framework

We've discussed a bit about how speed and accessibility can be challenging for Web Developers. I'm going to share a little bit more about other challenges and opportunities Web Developers face. The first big challenge is managing JavaScript frameworks and libraries. How developers leverage JavaScript has dramatically changed in the last ten years. These frameworks provide features and templates to make coding with JavaScript faster and easier, but there is some work required.

First, you have to make sure the architecture of your website matches the framework. Some frameworks like Angular are fairly strict, while React is pretty flexible. Both have their advantages and disadvantages depending on what you're trying to do.

Once you have a framework in place, you have to make sure you keep it up to date. Angular releases a large update every six months, but between the updates, the best practices can change.

The tools developers use have also seen significant changes in the last few years. Creating a development environment on a developer's computer is complex. Tools like Vagrant and Docker provide different ways of creating and managing these environments. Usually, a company will pick one tool and all the developers will learn how to use it.



Programming Blocks of a Website

2. Walk Through Resolving a Bug

Let me walk you through a recent bug I encountered. One of our customers reported the browser showed them '$NaN' when they tried to use our new feature for calculating the estimated cost of services. Hmm.

Since I wasn't there and didn't see this happen, I have to try to recreate it. The best first step for me is to ask the client for details about what happened. What buttons did they press, what were they doing, and in what order?

The reason this is helpful to me is that software is logic-based. Meaning, if this happens, then do this. So, if I

can understand the exact steps the customer took to make this bug occur, I have a better chance of making it happen myself in my local environment.

Use the following steps to resolve a bug:

1. **$NaN!:** The customer told me they selected a product marked at $150. The field indicating the estimated price showed $150. Then, they clicked the quantity dropdown and selected 2. Right after the estimated price showed $NaN! To a non-developer, this might not mean much. But to me, this is a breadcrumb I can follow. I know that piece of functionality takes a number value from our database and then multiplies that number by the quantity the customer inputs. So my next logical step is to go look at that code and see if I can spot the problem.
2. **Can I replicate?:** Once I looked at the code, I saw that the service cost is a number coming from the database. Maybe the data type was set incorrectly? Upon review though, it looks fine. The number is an integer as I expected. The next thing to do then is spin up my local dev environment and see if I can replicate. After selecting a product and multiplying by 2, I get the same as the client; '$NaN'! I haven't solved the bug yet, but I can replicate it consistently, and that's a good place to be.
3. **Do some logging:** The final step was adding in some logging. I want the computer to tell me what the value is at each step. When the value is first retrieved from the server, it's a number. But then, it looks like we are doing some formatting of that value by taking the number and turning it into a string so it can have the '$' character in front. It looks like that's the problem! The computer doesn't know how to multiply the number 2 by a string. I make a quick change to make sure that the multiplication uses a numeric value, and we're good to go.
4. **Testing, testing, testing:** Now I need to update my unit tests. Unit testing allows us to test the individual parts or components of software to make sure they work. It is like testing your car battery if your car won't start. The battery is just one part of a larger system that starts your car, but it is pretty easy to test individually to make sure it is working. Before I push my code live, I need to make sure my unit tests are set up to ensure that the multiplication function takes in the correct values and always returns a correctly formatted multiplied number. That's what testing does. It gives the team peace of mind because our tests make sure that the code we write acts the way we intended it to.
5. **Return $(a number):** In this case, I'm going to write a quick unit test and check for some returned values. I write the test using the test condition that multiplyProduct() should return a number formatted with a '$' character in front.
6. **Integration testing:** There is one more test to write. I want to make sure that the users will see the correct value as well. Unit testing is great for testing isolated pieces of code, but it doesn't test the software as a whole. I'm going to write an integration test now. Integration tests let you take multiple pieces of the program and combine them to make sure they all work together seamlessly. I know several steps the user must take to get to the point where they can see the multiplied values. So I will write an integration test that combines all those parts of the program, checking for correct values along the way.

One challenge you might not think about is how websites work in different browsers. A browser is a software application you use for accessing web content on the internet. All modern operating systems come with a browser already built-in.

You probably didn't realize that there are different options and that browsers can behave differently. Chrome is the most widely used browser currently, but Safari, Firefox, and Edge are also popular options. Edge is

Microsoft's new built-in browser that replaces Internet Explorer for better performance.

DID YOU KNOW

Developing websites to work in Internet Explorer was so notoriously painful that some companies decided not to support it.

Web development can be a great stepping stone to other IT careers. I'm really interested in server management, so I've been learning more about Linux, which is an open-source operating system used by most servers. Monique (Software Engineer) has shown me what she's learned and how she makes the servers scale when the traffic increases, but also how she shrinks it back down to manage costs. It's some pretty cool technology.

I've also been learning more about security best practices and how to code my sites to prevent malicious activity or hacking. It seems like a data breach is in the news all the time. I'm working to keep Poodle Jumper off that list by learning how hackers exploit vulnerabilities and preventing them wherever possible.

SUMMARY

This lesson discussed challenges that web developers face in **website architecture and framework**, as well as **resolving a bug** in their code. Developers must manage JavaScript frameworks and libraries and ensure that the architecture of their website matches the framework. They also need to keep their frameworks up to date and use tools like Vagrant and Docker to create and manage development environments. When resolving a bug, developers should recreate the issue, replicate it, do some logging, test the code, and write integration tests. Furthermore, developers need to ensure that their code works seamlessly across different browsers.

Source: This tutorial was authored by DEVMOUNTAIN and Sophia Learning. Please see our Terms of Use.