BRAD TRAVERSY'S WEB DEV GUIDE

A complete guide to web development technologies, concepts, career paths and more

2022 - 2023 VERSION

Brad Traversy's Web Dev Guide

© 2022, Traversy Media

Version 1.0 - May 2022

Table Of Contents

Introduction

About Me

I want to start this book off by introducing myself. Some of you reading this already know me, but for those that don't, my name is **Brad Traversy** and I started learning web development around 2007 at the age of 26. I won't go into my whole story, but I grew up with little money and I fell into drug addiction at the age of 17. I have been homeless, in rehabs and even spent over a year in jail for drug-related crimes. I found out I was going to be a father at 25 and I got clean and changed my life for the better. I found a passion for programming as well as teaching.

My back story is for another book (I do plan to write), but I just want you to know where I come from because I think it shows that web development is not only for the brainiacs that got straight A's and never left their computer as a kid. One of my missions is to bring as many people into this industry as possible.

Most of my career has been working as a freelancer and a contractor, but I have also worked for large companies and built my own products.

In addition to that, I started teaching and creating educational content around 2011. In 2016 I decided to do content full-time because, in addition to loving to write code, I have an even bigger passion to help others learn. I especially like helping people like me that struggled in life and people that you may not have expected to be a developer.

Why I Wrote This Book

Learning web development can be very overwhelming. I think that the biggest reason for that is because there is just so much to it and so many technologies. It's difficult just to **learn what you have to learn**. Languages, frameworks, libraries, plugins and the list goes on. It's enough to drive someone crazy!

I run a popular YouTube channel called Traversy Media and to address the problem that I just mentioned, every year I put out a **Practical Guide to Web Development** video. Many of you have probably seen this. The video is usually about an hour long and I go over as many web development-related technologies that I can, from basic tools to front-end and back-end frameworks to databases and web servers. I also do it in a linear order so that developers can use it as a map of what they should be learning.

Every year when I make this video, I script out everything and I end up having to cut it down quite a bit because I don't want the video to be too long. I always feel like there is so much more to say. So this year, I decided to create this eBook version so that I could go more in-depth and not only add more technologies, but talk about more concepts, and go over them more thoroughly. I also add more of my own experience & opinions and in some cases, I give some simple examples just to get you going.

I used this book to answer a lot of common questions that I get from viewers. It is essentially **a brain dump of everything that I know about web development technologies.** I will also talk about other things such as career paths, learning tips, trends, and more.

This is the **2022 - 2023** version, however there are new things coming out all of the time, so I will be adding to this guide every couple years.

Some Notes About This Guide

There are a couple of things that I want you to keep in mind as you read this book.

First off, I am not a “writer”. I don’t think I am horrible at it, but in no way am I a professional author. I could have had someone else write the book, but I wanted to write this the same way as if I were talking to you as a friend, much like I try to make my videos. I will not try and sound smarter than I am. There may even be some grammatical errors. I don’t care much about that. What I care about is having you understand what I’m trying to get across. Nobody else has written or edited anything in this guide aside from myself.

Second, the point of this guide is to help you understand the different parts of being a web developer and what types of technologies are available. I want you to be able to use this to strategically plan out your career path and know exactly what you should learn. **There is no one size fits all approach** to learning or choosing a path.

Keep in mind that the main point of this book is to let you know what everything is and what to learn. I teach some basic knowledge in certain sections, but there are full books written and full courses on each technology and the concepts I talk about. If you want to start learning from some of my content, I have put together a content guide that includes many of my YouTube crash courses and other videos from my channel as well as some of my Udemy courses in the same order of this eBook. You can find that at https://traversymedia.com/guide.

Try not to get too overwhelmed with the amount of information in this book. **There is no way that you're going to learn everything that we talk about**. It would be impossible. Pick your path and learn one thing at a time.

It's up to you how you read and take in the knowledge in this book. My suggestion would be to read the whole thing first and then go back and map out exactly what you want to learn and then learn it in a linear order. I will give some tips on how to learn in the next section.

I will be putting links to some of the websites for certain things throughout the book, but at the end, I will have a long list of links to everything that I mention in this guide.

CHAPTER 1

LEARNING & PLANNING

In this chapter we will talk about how to learn with some of my personal tips, as well as the different options and paths that you can take as a web developer.

First: Tips On How To Learn

Most of this guide is showing you “what” to learn, however in this section, I am going to talk about “how” to learn. This can be a difficult thing to explain or teach because everyone learns in different ways and at different levels. I will give you some general advice and also tell you what works for me. I will talk a little bit about formal education in the next section, but this guide is generally to help self-taught developers find their way.

Planning

Once you figure out what to start learning from this guide, make a clear ordered list or map of those topics. Just so you have some kind of plan. The more organized that you are, the easier time you will have.

Next, you want to pick one area/subject and focus on that for a period of time. I can’t tell you the exact period, because it depends on the subject, yourself and the time that you have available. Let’s say you are learning React. You may give yourself a month of studying 5-6 hours per day. If you have a full-time job or school, you probably don’t have that kind of time. So maybe do 2-3 months with 1-2 hours per day.

Everyone’s situation is different. There is no magic number for this. You’ll have to do some trial and error.

Learning Methods

You want to find good resources to learn from. This will depend on your preferred methods. I like to use video courses along with documentation. Here are some popular learning methods.

✔ YouTube (Usually shorter videos)

✔ Video Courses (Longer form videos eg. Udemy, PluralSight, etc)

✔ Traditional Books or eBooks

✔ Documentation

✔ Blog posts & Articles

✔ Podcasts

✔ Other People's Code (eg. Github)

✔ Formal Education (More in the next section)

I would suggest trying out a few methods and seeing what really works for you. As I said, I prefer video because I like to learn with real projects. So I like to build along with the instructor. This is also how I like to teach.

You should also find a few different resources and instructors that you like. You may like my way of teaching, but I would suggest finding one or two others as well because each instructor has a different point of view. If you are learning React, maybe purchase 2 or 3 React courses from multiple instructors if you can afford it. If not, YouTube has a lot too.

Immersing Yourself

I know that we live in an age of instant gratification and short-form content such as TikTok videos and YouTube shorts, but in my opinion, to really learn something, you have to get your mind out of that space.

What helped me really learn and retain information was to make my learning/study time almost like a meditation session. Set a good amount of time aside, and find a quiet place, preferably a clean and comfortable space. Grab a coffee or tea and really immerse yourself in the content that you are consuming. Make it peaceful and enjoyable. Try and look forward to it and never look at it as a chore.

Also, don't just mindlessly follow tutorials and copy code. If you don't fully understand something, pause it and look up whatever you are having trouble with. That's why I love to supplement video courses with documentation. If you can't find the answer there, StackOverflow is great or just a simple Google search.

I would also suggest creating your own projects that are based on what you learned in the course/tutorial/book. This will give you some real-life experience where you are debugging, looking stuff up, etc.

Forget About Remembering Code

One of the big misconceptions with coding in general is that you need to remember the entire programming language that you work with. Remembering code is fine and it is something that you will do naturally as time goes on, but you don't study to remember code, you study to remember concepts, algorithms, etc and to understand. If all you are doing is remembering coding syntax, that isn't very helpful, because the second one variable changes, you'll be lost. Instead of trying to memorize the exact characters, understand what the code is achieving.

When learning about loops, don't even bother remembering the characters to type. Remember how iteration works and what the purpose is. To run a sequence of instructions until a certain condition is reached. If you truly understand that, who cares about syntax, just look it up if you need to. Most programmers are “professional googlers”. There is syntax that I have had to look up for weeks or months. It’s ok though because I understood the underlying concept. This will also make it easy for you to learn multiple languages because they all use the same fundamentals. I will talk more about fundamentals in chapter 7.

I suggest keeping the documentation open for whatever you are working with. That is where you are going to find the stuff that isn’t stored in your brain. Don’t feel bad about not remembering syntax. It is completely normal. You may have to look the same thing up for months, but one day, it will just stick.

What About Formal Education?

This guide is definitely geared toward self-taught developers. However, I think it’s a good idea to talk a little bit about formal education. One of the most common questions that I get is “Do I need to go to college?”. That is a tough one to answer. Everyone learns in a different way and has different goals. If you take the “I” out of the question and make it “Do all developers need a college education?”, then the answer is **absolutely not.**

Personally, I took a few semesters in community college and dropped out. Nothing I do now came from college or any kind of schooling. It was all self-taught. I also know loads of successful self-taught developers.

Is college a waste of time?

No, I would not say that college is a waste of time, however, I do think for some people, time could be better spent. I think the biggest advantages that come from getting a degree are the following…

1. The social learning setting. Working with and around other people gives you a more professional learning experience that is similar to the real world and working with a team.
2. Having a set curriculum. Some people need that structure and can not learn on their own.
3. The credentials definitely help when looking for a job. You don’t NEED a degree, but it sure does help you get your foot in the door.

Notice that neither of those mention the actual education that you get. That’s because I honestly think that if you are well-disciplined, **you can learn much more on your own**. Most college curriculums are really out of date. You also have to spend time in classes that have nothing to do with what you actually want to do. On top of that, you get massive debt for many years.

What About Bootcamps?

In my opinion, coding bootcamps are something that can be really helpful. They tackle 2 of the 3 positive aspects of college that I listed above and have many other reasons why they are better than college (IMO), which I will list below. The issue with bootcamps is that you really need to dig to find the right one for you. There are a lot of “scammy” bootcamps that take your money and teach nothing of value and do not help with employment. **Be sure to do your research before attending any bootcamp.**

Pros of Bootcamps over College:

■ Money: Much cheaper than college.

■ TIme: You spend months instead of years.

■ They teach very relevant and useful technologies.

■ Many help you get a job after you graduate. Some will not even charge you until you get a job.

I have not attended any bootcamps, but from talking to students, I have heard good things about these ones

✔ Lambda School

✔ Fullstack Academy

✔ Devmountain

✔ Coding Dojo

✔ FlatIron School

The Bottom Line

Everyone is different, so the question of "should I go to college?" needs to be answered on a case by case basis and there are many factors to look at. I think for most people, self-teaching is a great route because of the amount of learning resources that are available. There is nothing you will learn in college that you couldn't find online. Many instructors online are better than college professors in my opinion. You just need to know what to learn. That's where this guide comes in.

Take everything that I say here with a grain of salt because I am only one person with one opinion. Deciding to go to college or not is a HUGE decision and should not be taken lightly,

Overall Goals

Before we go into specific technologies, let's first talk about your overall goal(s) and why you even want to code. If writing code interests you, I think one of the most important things to figure out early on is what you want to do with this skill. One of the things that I love most about coding is just how much you can do with it. You're also not bound to just one thing. In fact, I've done just about all of the goal options I'm about to show you at one point or another. This also is not all of the options available. Just some of the most common

Option #1 - Work as a developer at a company

I would say that this is the most common route that people take once they learn enough either on their own or when they graduate from college or a bootcamp. They may do something else later on, but this is usually the first big goal for many people.

Technology, software, and the web will be around for a VERY long time and will only become more important. This gives us a lot of job security, which makes being any kind of software developer a great profession. Not only that but web developers are paid pretty well.

Many developers are very passionate about coding, so to be able to get up every day and do what you love is a dream come true for many people. I think having the goal of being a high-paid developer at a good company is a great one to have. **It doesn't have to be a FAANG company** either. A lot of people seem to think you have to work at Amazon or Google to be a high-paid developer. That isn't true at all. There are thousands of large companies out there and even more small to medium-sized companies that have positions for web developers.

Some good websites to look for developer positions are

✔ indeed.com

✓ krop.com

✓ mashable.com

✓ careerbuilder.com

✓ authenticjobs.com

I will also include a link in the links section at the end of this book to an article that I wrote with 70+ job find websites.

Option #2 - Freelance or running your own business

I would say that freelancing is the second most popular route that people take when they learn to code. This was the route that I took. Nobody would hire me because I had a pretty rough history. That is for another book though. I decided to create my own business and it was tough getting started as it is for most people, however it ended up being very successful.

In order to freelance, you have to be very disciplined. This is true for any small business owner. You have no boss telling you what to do, which is what most people want when they start a business, however, you have to be your own boss and you have to work a little harder than you would at a company, at least in my opinion.

If you think that you have the discipline and drive to create your own business, this may be the route for you to take. I hate to throw in a promotion here, but it is extremely relevant. I created an entire freelance course with Kyle Prinsloo at https://freelancemastery.dev if you are looking to create a freelance business.

Some good websites to find freelance projects to get started

✔ upwork.com
✔ freelancer.com
✔ fivver.com
✔ guru.com

Option #3 - Build a product/SaaS

Building your own products is something that you can do full-time or while you are doing something else. When I was freelancing, I also had a side business selling WordPress & Joomla plugins. It gave me a nice little side income. I had other adsense websites as well as a website that sold PLR article packages.

Building a SaaS (Software as a Service) is something that is very popular these days. These are typically products that you can license out or charge a recurring fee for. The possibilities are endless, but just to give you some examples of things to build…

- Office Software
- Messaging Software
- Content Management Systems
- Selling Plugins, Templates, Designs, etc
- Hosting Solutions
- Social Media Tools
- Talent Acquisition
- Geographic Information Systems
- Virtualization
- DBMS Software

The list just goes on and on. Yes, many people fail when they build these projects, whether it is because they didn't have good marketing or maybe the product just wasn't that great. Either way, you will have failures no matter what you do, but the road to success is paved with many failures.

Option #4 - Create Educational Content

Another option is creating educational content, which is where I ended up after doing just about all of the others. I started this not because I failed at the other endeavors. In fact, it was quite the opposite, but because I realized that teaching and helping people is what I loved more than anything. I also found that I had a pretty good knack for it. I will say that this is not usually the first route that people take. It is a bit strange to go right to teach without having experience in one or more of these other options.

If you want to get into content, I would suggest first doing free content. This is a ton of work without any financial benefit, but it gives you a chance to build up your name and reputation as well as practice and become a better developer and teacher. You can start with a YouTube channel and build an audience. Eventually, after you create some great free stuff, you can start creating premium courses or books or whatever you want.

Again, I would suggest not starting with this, but do it in your free time, while you work as a developer and gain some real-life experience.

Option #5 - Code as a hobby

A lot of people start out coding as a hobby. It is fine to keep it that way. You don't have to make a career out of it. Like I said earlier, it is something that a lot of people are passionate about and sometimes people like to keep their passion and their career separate. You also might start off as a hobby and come up with a really cool idea that makes you a ton of money someday.

You are not stuck with just one of these options. You can do more than one at the same time. I certainly have. It's also important to mention that what you learn may depend on what you plan to do. For instance, WordPress may be something that you want to learn as a freelancer, but if you're looking to work at a FANNG company, you probably won't be looking to learn WordPress. We'll talk some more about this later.

Types Of Web Developers

A web developer is someone who builds and maintains websites and web applications. They focus on things like programming, architecture, application integration, and even design and graphics. There are mainly 3 types of web developers and that is **front-end, back-end**, and **full-stack developers**. Next, I will briefly go over each type and what it typically includes.

Front-End Developer

The front-end of a website is the part that the user views and interacts with. So a front-end developer will create the user interface and be in charge of things like navigation, styling, UI components, state, etc. The most common skills that a front-end developer has to have are HTML, CSS, and JavaScript. Each has a very important role.

HTML - The markup of the page. Headings, paragraphs, links, etc
CSS - The Styling of the page
JavaScript - The dynamic interactivity & logic

Front-end devs will also typically use a front-end JavaScript framework like React or Vue, which allows them to create really in-depth interfaces in a much easier way than if they just used vanilla JavaScript. We'll go over frameworks in a bit.

They also typically work with APIs (Application Programming Interfaces) that back-end developers create so that they can display and manipulate data from the backend server from the UI. This data is usually in JSON format.

When it comes to the actual design, it really depends on the company. Many companies will often have designers create everything visually and then the front-end developer will take the design and create it as a functioning website. For smaller companies and freelancers, sometimes the developer is also involved in the design process.

Average Salary

According to glassdoor.com, the average salary of a Front-end web developer in the US is around **$93,167** per year and base pay of **$80,513** per year**.** Of course, factors like education, experience, certifications, and more will affect this number.

Back-End Developer

You can think of a web application as a house. The front-end developer does the painting and interior designing and placement. The back-end developer does the foundation and the walls that hold everything up.

Back-end developers create the functionality and logic that runs on the server. The browser on the client's machine loads the HTML, CSS, and JavaScript but the server could be any language including PHP, Python, C#, and even JavaScript (Node.js).
If you have an eCommerce website, the front-end developer creates the buttons and forms, etc but the actual ability to place an order happens on the back-end. They work with the actual data, usually stored in a database. So back-end devs usually need to know a programming language as well as a database system such as Postgres, MySQL, or MongoDB, which we will talk about later. They interact with the database through their code using different tools and libraries.

Back-end devs are not usually concerned with what a website looks like at all. Their job is to make sure the server code is serving data, processing data, etc. They will typically build APIs for the front-end developers to consume and work with.

Average Salary

According to glassdoor.com, the average salary for a Back-end developer in the US is around **$92,238** per year and base pay of **$74,096** per year.

Full Stack Developer

A full stack developer does what you would expect and works on both the front-end and back-end. Depending on the job and/or the project, they usually work more on one end or the other, but you should pick a stack and study everything from creating APIs to creating UIs if you are looking to be a full stack developer.

It's also important to mention that these days the lines that separate the front-end and back-end are becoming more and more blurred with frameworks like *Next.js* and we'll talk more about those later. This guide is for everyone, but I wrote it mostly with full-stack devs in mind because this seems to be the most popular choice, especially with my audience. I also see a lot of people start off as front-end or back-end developers and move to full-stack later on.

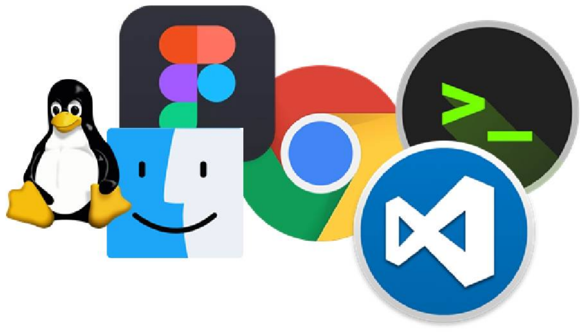
Average Salary

According to glassdoor.com, the average salary for a Full-stack developer is **$109,136** per year and base pay of **$95,380** per year. Naturally, full-stack devs will make more because they need to know more.

CHAPTER 2

FRONT-END BASICS

In this chapter we will cover what it takes to get started and how to become a basic front-end developer. This includes the basic software tools that you need and web technologies like HTML, CSS and JavaScript.



Tools: The Necessities

Now that we have talked about the different options that you have and the different types of web developer roles, let's start to look at specific technologies and topics. We will start with the absolute basics of what every web developer needs.

Computer & OS

Obviously you need a computer to write code and you need an operating system on that computer. As far as the computer, you don't need a super-powerful machine. Anything made within the last ten years is fine for basic web development. I would suggest at least 8GB of memory though.

As far as the operating system, everyone has their own preference. There is no "best" operating system. I prefer MacOS, but I also have a

Windows machine that I use quite a bit and I also run Linux on my servers.

Like with most things in tech, you'll come to find that each operating system has its own group of "fanboys" that will make you feel like you're an idiot for using anything different. Try not to listen to these people. They're both ignorant and arrogant. Use whatever feels right for you.

I will say that there are a few types of developers that usually stick to a certain OS. For instance, C# was created by Microsoft and it is Microsoft that provides the runtime environment required for the operation of C# programs. So I would say that most C# developers run Windows. However, it is possible to develop C# on Mac or Linux. Another example is iOS developers. They typically develop on macOS because of the tools available. Other than cases like that, it comes down to preference.

Web Browser

A web browser is obviously something that you will need as a web developer. As far as which one, like your operating system, this is based on preference. I personally like Google Chrome. I have used it for the past decade or so and I like the developer tools that it includes, but these days, all modern web browsers have these tools to one extent or another.

The popular and modern web browsers that you have to choose from are the following…

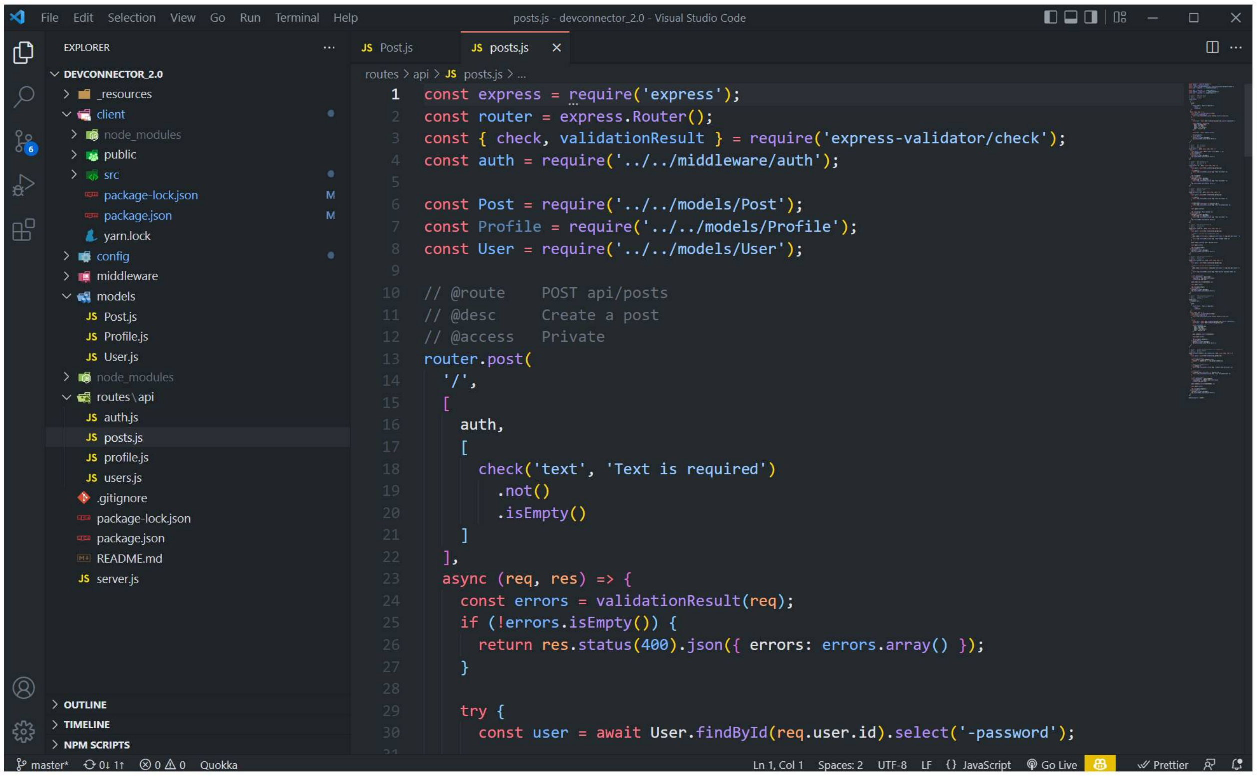
- Google Chrome
- Mozilla Firefox
- Brave
- Microsoft Edge
- Safari
- Opera

Text Editor / IDE

Text editors and IDEs (Integrated Development Environment) are used to write code. IDEs typically offer more features such as built-in compilers but are also larger, slower, and more difficult to use than text editors. Many text editors, such as VS Code are very full-featured and offer all that you would need for most web development code. They also offer extensions to add more functionality. Let’s look at some popular options for web development.

Visual Studio Code

Microsoft's VS Code is my text editor of choice and is extremely popular for all types of web development. It is open-source and free. It has a ton of great extensions and can be used for HTML, CSS, JS, Python, PHP, and just about any other language. If you want to see the difference between a text editor and an IDE, "Visual Studio" is a complete IDE. VS Code is a text editor that was based on that, but much lighter and faster to work with.



```
const express = require('express');
const router = express.Router();
const { check, validationResult } = require('express-validator/check');
const auth = require('../../middleware/auth');

const Post = require('../../models/Post');
const Profile = require('../../models/Profile');
const User = require('../../models/User');

// @route    POST api/posts
// @desc     Create a post
// @access   Private
router.post(
  '/',
  [
    auth,
    [
      check('text', 'Text is required')
        .not()
        .isEmpty()
    ]
  ],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    try {
      const user = await User.findById(req.user.id).select('-password');
```

Sublime Text

Sublime Text is another great text editor that is extremely lightweight and fast. I used it for about two years and still use it from time to time. In fact, since it is so lightweight, I use it for reading and viewing .txt files as a replacement for Notepad. Technically, it isn't free, but the trial never ends. You just get an annoying popup every now and again. If you use it, I would suggest paying for it. It also has a plugin system and a lot of great features, but it is a bit more difficult and less intuitive than VS Code in my opinion.

Atom

Atom is an editor that is pretty similar to VS Code in features, extensions, etc. I haven't used it in a while, but it was heavier than VS Code in terms of memory usage, etc. I can't say if they have since fixed those issues. I still see quite a few developers using it.

Vim

Vim is what I like to call the "editor for nerds". I don't mean that in a bad way, it's just the people that use it are very passionate about it. There is nothing wrong with being a nerd. Vim is the successor of Vi, which is a

Unix-based text editor. Vim can be used as a CLI (Command Line Interface) or a standalone GUI (Graphical User Interface) and is extensible and customizable. It is definitely the most difficult, in my opinion, and there are a lot of different hotkeys to remember.

Terminal

The terminal or the command line is a text-based tool to navigate your computer system and run programs. Every Operating system has some kind of terminal. You can also install 3rd party terminal software. This is not something that you need to know before you get into web development, but it is good to know of and know that you will need to learn at least some basic commands to navigate your system. You can just use the default OS option, but here are some others.

■ **Powershell** - A more advanced version of the default Windows cmd

■ **iTerm2** - 3rd party terminal for Mac users

■ **Hyper** - 3rd party, cross-platform terminal with plugins

■ **WSL (Windows Subsystem for Linux)** - Allows you to run Linux on Windows

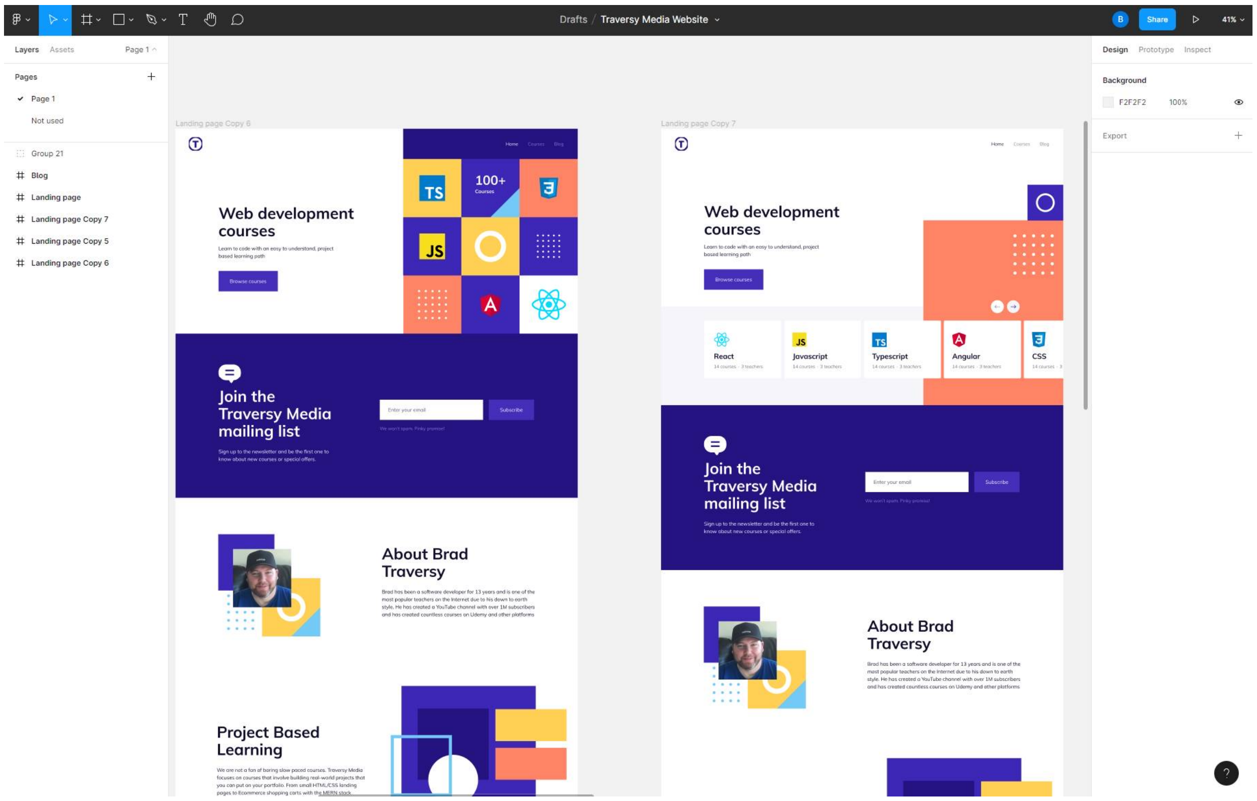
■ **Git Bash -** 3rd part Unix based terminal for Windows.

Design Software

A lot of new developers ask me if they need to learn design software. As I stated in the front-end developer section, it really depends. You may work for a large company that has designers that hand you a mockup to turn into a website/application UI. You may also work for a small firm or freelance and end up doing all the design work. So I would say at least learn the basics of one of these programs.

Figma

Figma is extremely popular right now. Designers love it, front-end devs love it. It is free and runs right in the browser. It is extremely full-featured and you can build some great stuff including interactive mockups. In addition, it is very collaborative and easy to share and work with other people. I would probably suggest Figma to most front-end developers and UI/UX designers.



Adobe XD

Adobe XD is my personal choice. I have always been a fan of Adobe products. I spent 7+ years freelancing and for many of those years I had to do all the design myself. I used Photoshop for mockups, which worked well, but I moved to XD when it was released because it is more focused on creating UIs than photo editing. If you like Adobe products, then this may be for you. Adobe XD alone is $9.99 per month or you can get all of the Adobe products for $52.99/mo

Sketch

Sketch is a great tool for UI/UX. If you are on a team, it has a lot of collaboration features. Sketch runs on the Mac platform, but I believe there is a web app platform that runs in the browser. The standard plan is $9 per month right now.

Photoshop

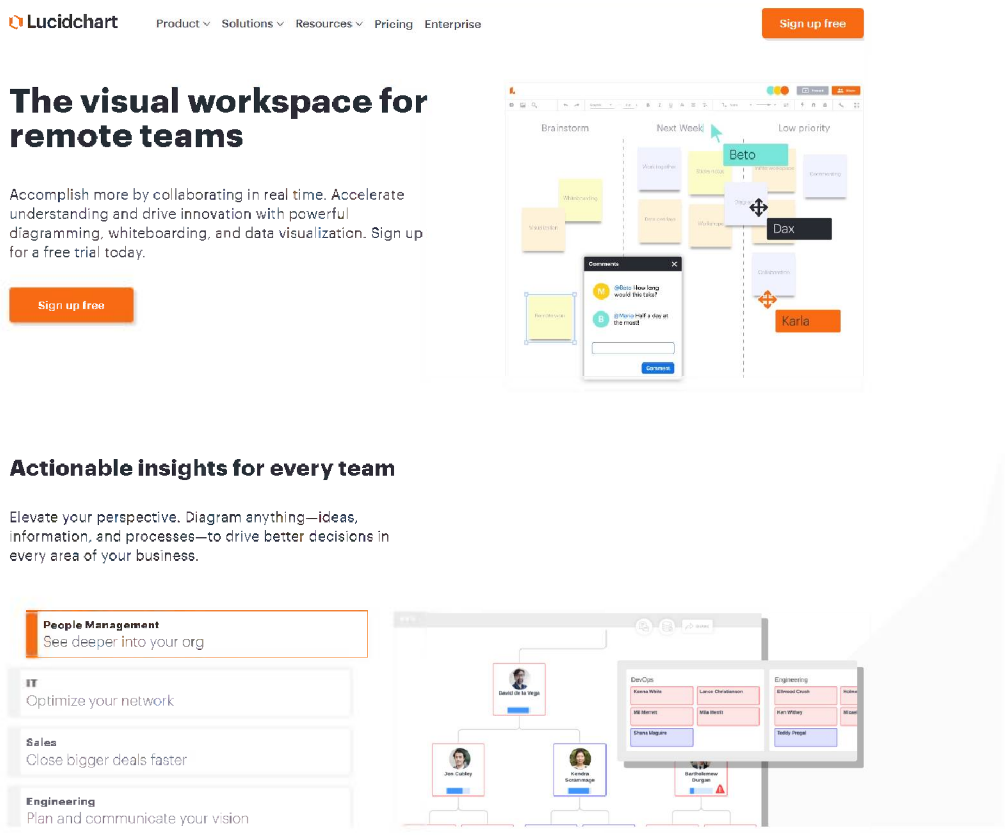
Even though I would personally suggest Adobe XD over Photoshop, it is still a viable option. Although it is geared toward photo editing, it can be used to create mockups and UIs. I used it for years to create mockups for clients.

Canva

Canva is a great browser-based design tool. It is used for everything from business cards to presentations to website mockups. There is a free individual plan as well as paid plans for teams.

There are many others as well. You usually do not need to know everything about the software. I would say that I only know about 35% - 40% of Adobe XD features and maybe 25% of Photoshop, and I used both to create professional mockups. This also is not something you need to know right now. I would just pick one, install it and move on for now. The time will come when you need to start slicing up images.

There is also wireframing software that is great for just laying out the content of your website/app. They usually don't have a ton of design elements, just a basic diagram. Something like **Lucidchart** or **Invision**.



The visual workspace for remote teams

Accomplish more by collaborating in real time. Accelerate understanding and drive innovation with powerful diagramming, whiteboarding, and data visualization. Sign up for a free trial today.

Sign up free

Actionable insights for every team

Elevate your perspective. Diagram anything—ideas, information, and processes—to drive better decisions in every area of your business.

People Management
See deeper into your org

IT
Optimize your network

Sales
Close bigger deals faster

Engineering
Plan and communicate your vision

HTML & CSS

Now that we have gone over the basic tools, we can start to get into web technologies.

HTML & CSS are always the first things to learn when getting into web development because these are the building blocks of the web. No matter how sophisticated your web app, no matter what language is used on the server or which front-end framework you're using, in the end, it spits out HTML which stands for **HyperText Markup Language** and is basically the skeleton of the web page. CSS, which stands for **Cascading Style Sheets, is used** for the styling of that web page.

Typically, you're going to learn both of these at the same time but you'll be spending most of your time with CSS because it's much more difficult and there's just much more to learn than with HTML.

I realize that most of you reading this probably at least know the basics of HTML & CSS. If not, I have crash courses on YouTube for both.

HTML Basic Tags & Attributes

For many people, the very first thing that we do when we start learning web development is create a .html file and start writing HTML tags. Learning how to structure a document with the <head> and <body> tags and then learning all of the basic tags like paragraphs <p>, headings <h1> - <h6>, Lists <ul> <ul> and the rest of the common tags.

You will also learn about block-level vs. inline elements as well as all of the attributes that go along with certain tags such as "href" on links <a> and "src" on images <img />.

HTML5 Semantic Tags

You also want to learn how to layout your pages with HTML5 semantic tags like header, footer, section, main, aside, and so on. Before HTML5, we basically used <div> tags for everything and depended on IDs for naming elements. Here are the most commonly used semantic tags for layout. They are pretty self-explanatory on what you would use them for.

<header>

<footer>

<nav>

<section>

<main>

<article>

<aside>

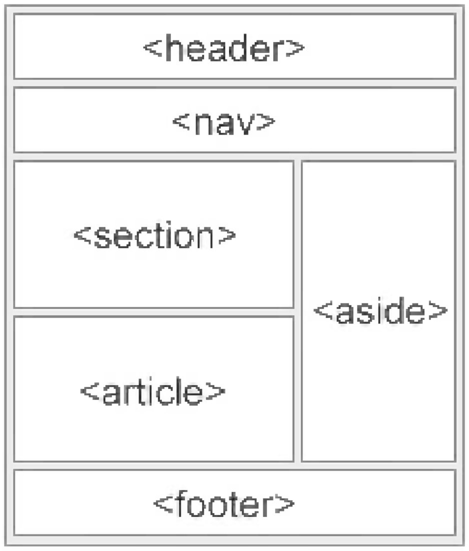


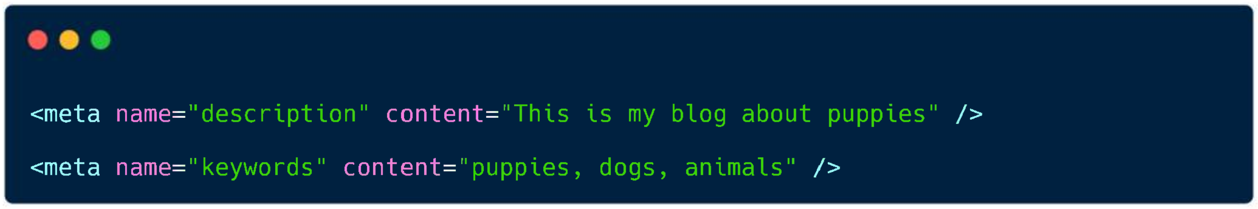
Image from w3schools.com

Above is an image of how your layout may be structured. Remember HTML is just the markup, so it has nothing to do with positioning and physical layout. That is the job of CSS, which we will talk about soon.

In addition to layout, forms are important because that is typically how we collect input and get data to the back-end. Learn all of the different input types as well as the attributes for stuff like validation.

Meta Tags & SEO

SEO is a big part of having a successful website. You don't have to be a master at SEO. That is usually a completely separate job or service. However, you should know the basics, especially freelancers. This includes certain meta tags. You can add <meta /> tags such as keywords and descriptions. These will help the search engines know what the page is about as well as know what to display in the search results. Here is an example of a description meta tag. They typically have a **name** and **content** attribute.



```
<meta name="description" content="This is my blog about puppies" />
<meta name="keywords" content="puppies, dogs, animals" />
```

The <title> tag is very important for SEO. Keywords and descriptions do not hold much weight anymore in terms of where your site ranks, but page titles do, so be sure to use short but descriptive titles.

Video & Audio Tags

HTML5 also offers a <video> and <audio> tag. You can simply add a url to a video or audio file and embed them, but they also come with JavaScript APIs to create your own players, which I'll talk about later. Here is an example to embed a video.

```
<video controls>
  <source src="mymovie.mp4" type="video/mp4">
  <p>Your browser doesn't support HTML5 video. Here is a <a href="mymovie.mp4">link
to the video</a> instead.</p>
</video>
```

The code above would display the video right on the page unless you were using a browser that did not support it, then it would show a link to the video itself. The controls attribute will show the play, pause, etc. Here is an example of the audio tag.

```
<audio controls>
  <source src="mysong.mp3" type="audio/mpeg">
  Your browser does not support the audio tag.
</audio>
```

CSS Fundamentals

Once you know how to layout your pages semantically and you understand the common HTML tags and attributes, now you will move on to CSS to style your pages. You want to learn some of the common properties that have to do with the following.

✔ Colors and borders
✔ Text sizing and alignment
✔ The Box Model
✔ Margin and padding
✔ List styling
✔ Form styling
✔ Borders & shadows

CSS Layout Options

When we talk about layout with HTML, we are strictly talking about the content layout and which tags to use as opposed to the physical layout. One of the most important parts of CSS is how you layout your elements. There are a few ways to create a layout with CSS.

Floats

When I got started, we only had floats to work with in order to have rows of content. Float is a CSS property where you can set an item to float left or right. It was not intuitive. Sometimes people even used tables, which were a nightmare. Personally, I don't think that you really need to learn floats now. If you really want to, just learn what floats do and move on.

Flexbox

Flexbox is one of the best things ever to come to CSS. It allows you to easily create one-dimensional rows with columns of content. So you can easily create navigation menus, rows of cards, etc. We use flexbox by having a **flex containe**r and then multiple **flex items**. Something like this…

```html
<div class="my-flex-container">
   <div class="flex-item-1">Item 1</div>
   <div class="flex-item-2">Item 2</div>
   <div class="flex-item-3">Item 3</div>
</div>
```

On the flex container element, in CSS, we would add **flex** as the display property like this…

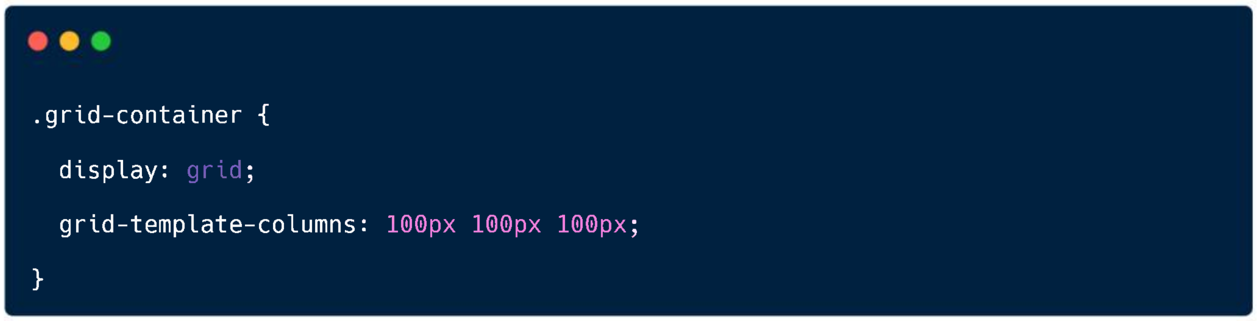
```css
.my-flex-container {
   display: flex;
}
```

Then all of the flex-item-x divs would be treated as flex items and we could arrange them however we want into rows (horizontal) or columns (vertical) using CSS properties that are specific to flexbox and layout like **align-items** and **justify-content**. My Flexbox crash course on YouTube walks you through all of this.

CSS Grid

CSS grid was released a little while after flexbox. It allows you to create two-dimensional grids. Unlike flexbox, you have full control of columns as well as rows.

It is used in a similar way to flexbox as you set your display property to **grid**. Then you can set the number of columns and/or rows that you would like to use. Your CSS would look something like this.



```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
}
```

So here you would have a 3 column layout with each column being 100px each. More commonly than using “px” units, you will use “fr”, which are fractions.

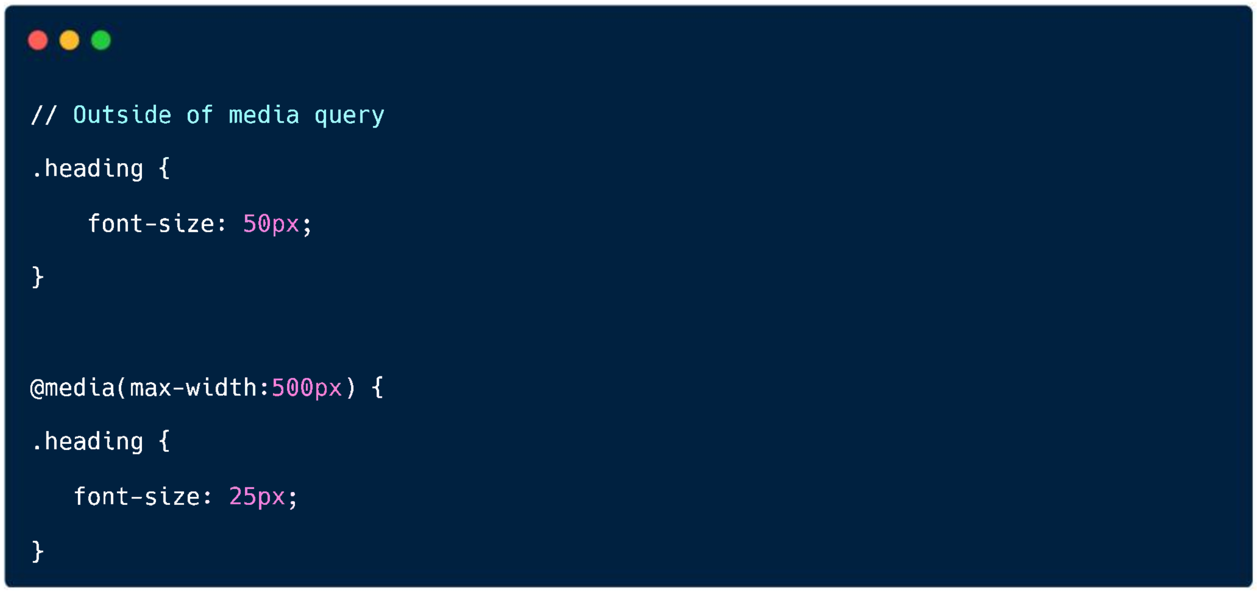
I would suggest learning both flex and grid because they can be useful in different ways. Personally, I like to use grids for my overall layout of content and then use flexbox to align inner elements on the page. You will find your own groove though.

Media Queries & Responsive Design

At this point in time, anything that you create for the browser should be at the very least, decent looking and readable on mobile devices. Five to ten years ago, responsive design was an added feature, but now it's needed.

There are a few things that go into responsive design such as setting the viewport and using flexible widths. The main ingredient though, are CSS media queries. They allow you to write specific CSS for specific screen sizes.

For example, you may have a heading with a font size of 50px. Maybe on mobile devices (around 500px max), you want the size to drop down to 25px. Here is how you would write this as a media query.



```
// Outside of media query
.heading {
    font-size: 50px;
}

@media(max-width:500px) {
.heading {
   font-size: 25px;
}
```

If you use flex and/or grid, you have multiple columns. On mobile screens, you do not want 3 or 4 columns because it would be very crowded. Typically, you would create a media query to set the layout back to a single column on small screens. This is called a “stacked” layout.

Mobile First Design

When you create a responsive website, you have the option to design **desktop-first** or **mobile-first**. It is pretty self explanatory what this means. Which layout do you want to start with? I prefer to build desktop-first unless I expect more traffic from mobile users. Many developers are taking a mobile-first approach. Even frameworks like Tailwind and Bootstrap are “mobile-first frameworks”.

Progressive Enhancement is a buzzword we hear a lot of. Basically we start with a more simple layout with basic functions and features and then we tend to add more and enhance the design for tablets and PCs.

Graceful Degradation is basically the opposite. Where we start with the more advanced design and then take things away for smaller screen sizes. This is how I usually prefer to work when creating layouts.

Simple Animations & Transitions

Simple animations and CSS transitions are not crucial at this time, but I would suggest at least learning the basics. When I first got into web development, jQuery was really big and we did a lot of the simple animations with it. These days, it is usually done with CSS transitions, keyframes, etc. Once you learn JavaScript, you can incorporate that as well.

Moving On

Once you can build a fairly decent-looking website and understand the fundamentals of HTML/CSS, then you are ready to move on to learning something else. However, this does not mean that you are 100% done learning HTML/CSS. You'll learn more as you complete more projects.

You'll get better at design as you progress. You'll work with many different types of websites and even though you are not a "designer", you learn to know what looks good and what doesn't. Many of it is subjective, but some things just objectively look good or bad. Again, if you are planning on freelancing or working for/starting a small agency, you will need to understand design better than a front-end dev at Google, because you're probably going to do everything yourself at first.

PSD to HTML Gigs

A lot of people look for this type of thing. They are a designer or have a designer create a PSD (or other design file) and hire people like you to create a functioning website from it. PSD to HTML were some of my first gigs as a freelancer. I would probably say to hold off at this point, but If you are ready to work, then search some freelance sites for these types of jobs.

Sass

Sass is a **CSS preprocessor**. It is basically CSS with some added features such as variables, nesting, functions, modules and operators.

You use Sass by creating a **.sass** or **.scss** file, writing your Sass code, and then using a **Sass compiler** to compile it down to regular CSS to put in your project. There are both command-line and GUI compilers.

I do think that Sass will be phased out over the next few years because CSS has really stepped up its game. We now have custom properties in CSS, which are basically variables. CSS will also have nesting soon as well. With that said, **I** do still think that you should learn the basics of Sass because it is relatively easy to learn. If you already know CSS, you can literally learn it in a weekend. I also think that you will run into it at some point so it is good to know.

We are going to talk about CSS frameworks next but another reason to learn Sass is that if you want to customize Bootstrap, for example, you will be dealing with Sass files to edit certain variables and then recompile it into a custom Bootstrap file.

Let's take a quick look at the basic Sass syntax and how some of the features work.

In Sass, we can create variables like this and reference them anywhere.

```scss
$primary-color: #333;
$secondary-color: #444;
```

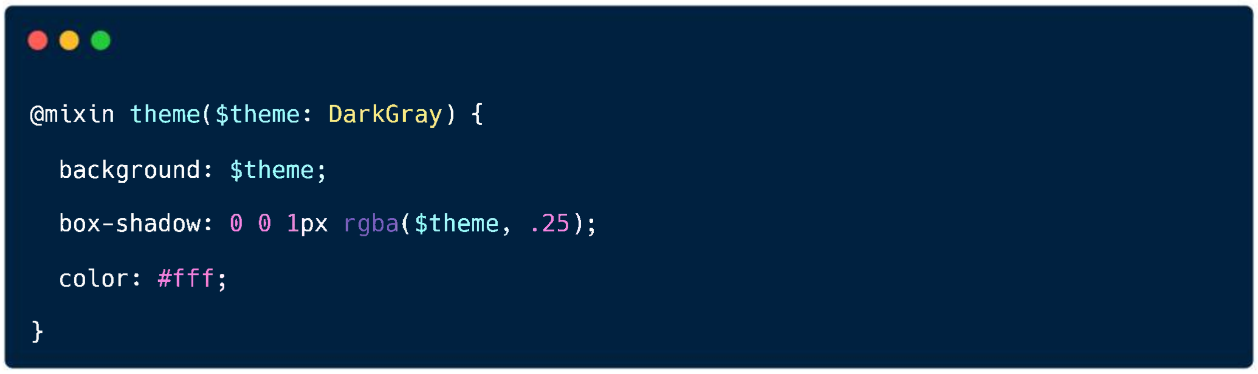
We nest styles like this.

```scss
nav {
    height: 50px;

    ul {
      list-style: none;
    }
}
```

This would only give the list-style: none to uls inside of a nav tag.

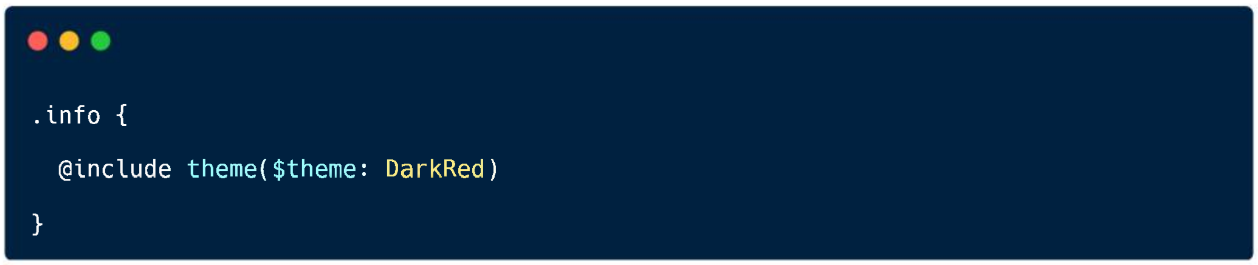
We can also use something called mixins, which let you make groups of CSS declarations. They are similar to functions. They can even take in arguments.

Let's take a look at a simple mixin.



```
@mixin theme($theme: DarkGray) {
  background: $theme;
  box-shadow: 0 0 1px rgba($theme, .25);
  color: #fff;
}
```

To use a mixin, you use the @include directive.

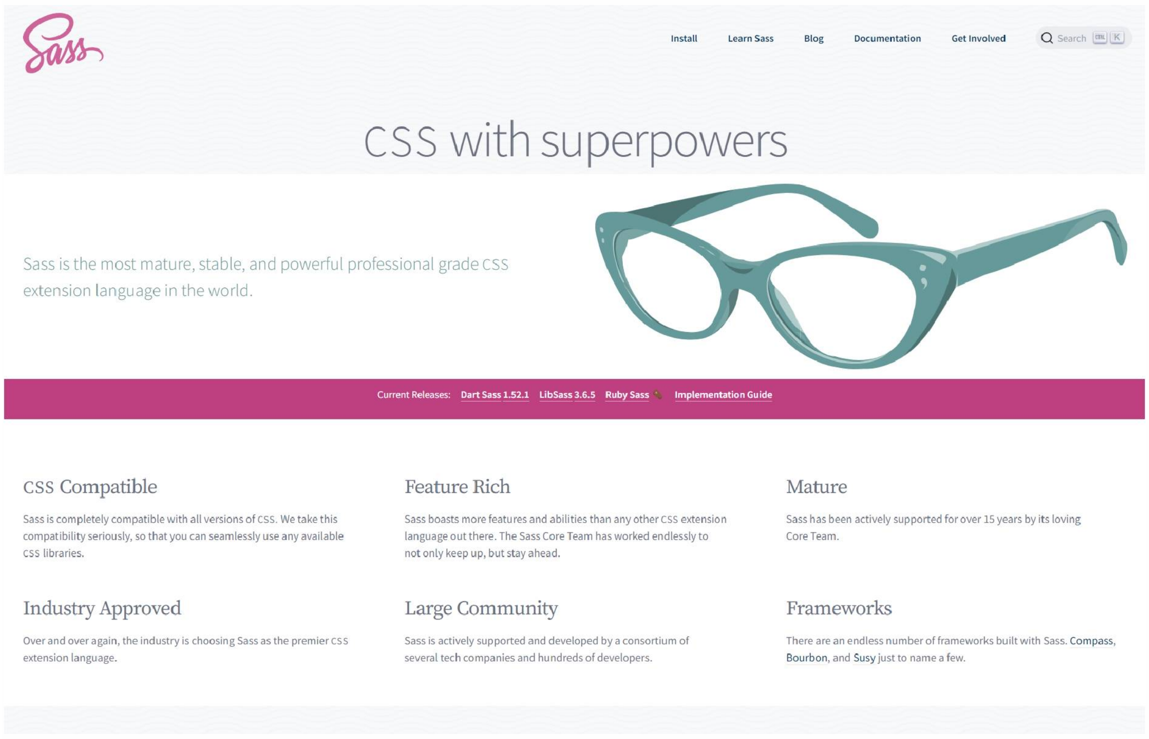


```
.info {
  @include theme($theme: DarkRed)
}
```

You can also share a set of CSS properties with another using inheritance.

```scss
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.message {
  @extend %message-shared;
}
```

There are more features to Sass. If you are interested, check out https://sass-lang.com. I also have a Sass crash course on YouTube.



CSS with superpowers

Sass is the most mature, stable, and powerful professional grade CSS extension language in the world.

Current Releases: Dart Sass 1.52.1 LibSass 3.6.5 Ruby Sass Implementation Guide

CSS Compatible

Sass is completely compatible with all versions of CSS. We take this compatibility seriously, so that you can seamlessly use any available CSS libraries.

Feature Rich

Sass boasts more features and abilities than any other CSS extension language out there. The Sass Core Team has worked endlessly to not only keep up, but stay ahead.

Mature

Sass has been actively supported for over 15 years by its loving Core Team.

Industry Approved

Over and over again, the industry is choosing Sass as the premier CSS extension language.

Large Community

Sass is actively supported and developed by a consortium of several tech companies and hundreds of developers.

Frameworks

There are an endless number of frameworks built with Sass. Compass, Bourbon, and Susy just to name a few.

CSS Frameworks

So before you go on to learn JavaScript, you have the option to learn a CSS framework. One thing that I really want to mention that is extremely important in my opinion, is that you should **fully learn CSS before jumping into a framework**. I have seen a lot of people jump into frameworks too quickly and all they are doing is memorizing classes and not really understanding the CSS. This really limits what they can build.

Once you can create a decent-looking website with just HTML and CSS, you may want to look into learning a framework because they can really help you speed up your projects and minimize the amount of CSS that you write. In some cases not writing any CSS at all. CSS frameworks give you predefined classes to use in your HTML rather than you writing the CSS. There are some downsides including less flexibility and sometimes websites can look very similar when they use the same framework.

My personal opinion is that you should learn one of these because learning the basics is pretty easy and really just consists of remembering some class names. Not even remembering, but just knowing how to look them up. Some frameworks also have some really useful JavaScript widgets.

Frameworks can really speed up your project's development time, but even if you don't use them in production, they can be good for development. If you are working on the back-end of a full-stack app and you just need something quick to show on the front-end, these frameworks are great for at least helping it not look absolutely horrible with no CSS at all.

So there are quite a few reasons to learn a CSS framework. You also don't have to learn one right this second. You can learn it later on.

Bootstrap

Bootstrap has been around for a long time and is currently on version 5. I would say that it's the most popular to date. I think it's worth learning for the simple fact that you'll probably run into it at one time or another.

Also, if you learn Bootstrap and you understand how it works, then you pretty much understand how all CSS frameworks work (aside from maybe Tailwind) and if you feel like using Bulma or something else for a project you can do that.

In addition to CSS classes, there are also a bunch of JavaScript widgets like hamburger menus and accordions. These can be tricky to create from scratch if you are not really comfortable with javaScript.

Tailwind CSS

Tailwind is a bit different than the rest. Tailwind is a low-level, utility-first framework. Let me explain what that means. Bootstrap is “high-level”, meaning that we can use a class of “alert” or “button” and we’ll get CSS styling with the background color, the border, border-radius, etc. In many cases, a class = a component (eg. alert, card, navbar).

What Tailwind does is provide “low-level” utility-based classes that usually do a single thing or add a single CSS property. For instance, the class of “text-center” will add text-align: center. The class “mr-1” will add a margin on the right to a specific size. The class of “ml-2” will have a little more margin but on the left. So instead of having component classes, you use utilities to create your own components. This allows for much more unique layouts.

One possible downside to using Tailwind is that you can end up having a lot more classes in your HTML, but the benefit is that you can customize things better and not all the buttons and alerts look the same as every other website. So there are pros and cons. You can also utilize directives like @apply in Tailwind and create custom classes to cut the amount you have in your HTML down a bit.

Tailwind may seem a bit annoying at first because of the number of classes, but it really grew on me and is my favorite CSS framework. It allows you to create layouts and change and test things extremely quickly once you get to really know all of the classes. It is also the most customizable framework by far. There is a Tailwind CLI that you can use to create a nice customizable environment. You can even add config values when you use the CDN.

Materialize CSS

Materialize is another framework similar to Bootstrap. It has CSS classes as well as JavaScript widgets and it's based on the Material Design

design pattern. I've always liked the look of Materialize, but if I were to recommend a framework to a beginner, it would probably be Bootstrap.

Bulma

Bulma is another nice-looking framework with CSS components. It is really modular, it's also mobile-first like most of these other ones. Bulma does not have any JavaScript widgets or anything like that. It is just a nice, lightweight CSS framework.

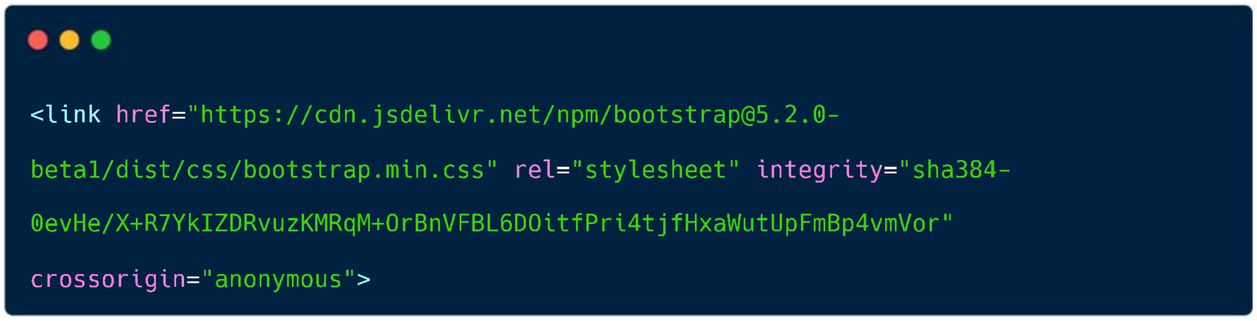
Foundation

Foundation is a well-established, full-featured UI framework that has been around for quite a while. It is still being used, even by some pretty large companies. It is a bit more advanced in tooling, but it offers a lot and it is very customizable. I stopped using it because I personally think some of the other frameworks have better-looking UI components.

Development Environments

CDN

There are a few different ways to use a CSS framework. The easiest way is to use a CDN link. CDN stands for **Content Delivery Network** and is used to distribute content from an “origin” server throughout the world by caching content close to where each end user is accessing the Internet. Most frameworks simply have a CDN link that you can pop right in your HTML <head> tag and you’re good to go. For instance, bootstrap’s CDN looks like this.



```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
0evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjfHxaWutUpFmBp4vmVor"
crossorigin="anonymous">
```

All you need to do to use Bootstrap classes is put that in your <head> tag

Custom Workflow

A CDN is great for small projects and learning, but in a real production site, you probably want to include the actual CSS file in your project structure. For one thing, you can’t do any customization. The bootstrap button will look just as it does on a million other sites. Also, the user has to be connected to the Internet for the CSS to load.

If you want to customize your layouts, you need some kind of local dev environment. It really depends on the framework. Tailwind has the Tailwind CLI, which I absolutely love. It makes it easy to customize your styling using directives and functions. It basically watches your html for Tailwind classes as well as any custom CSS you have and it compiles a custom stylesheet for you to include in your production site. Tailwind can also be installed as a Post CSS plugin if you want to use it with a build tool like Webpack. That is more advanced than where we are at right now though.

Bootstrap is a bit different. It uses Sass. So you can change some Sass variables such as the colors, button sizes, etc and then re-compile it into a custom bootstrap file for production.

So each framework is different. You could even create your own framework. If you learn Sass, you can easily have separate modules for buttons and navigation, forms and so on and then compile it into a single CSS file and call it a framework.

UI Design Principles

UI Design is something that I think is important to understand as a front-end web developer or web designer. As I stated earlier when we talked about design software, If you're working at a large company, they may have designers that create mockups, and then your job is to take that mockup and make it display and work in a browser. However, there are some companies where front-end developers do have some input on the design.

If you're freelancing, unless you plan on outsourcing all of your design work, you'll be creating the layouts. So **you should at least have somewhat of an eye for design** and understand some general principles. These are some of the really important ones...

Color Contrast

Color contrast is the difference in light between the font or anything in the foreground and the background. You want to make sure that the text is readable. We've all seen those websites where there's a large image behind the text with no overlay and you can barely read the text. You want to be aware of this if you are creating any kind of design.

Whitespace

Whitespace, also called "negative space" is the blank or empty space between elements on the page. It is important that you don't cram everything into a small amount of space. Use the right amount of margin and padding on your elements. You also don't want too much whitespace, where it feels like the page is too empty.

Scale

Scale refers to the relative size of one element compared to another. If you have a heading and a paragraph under it, make sure that the heading is not too big or too small relative to the paragraph.

Visual Hierarchy

Visual hierarchy is used to rank design elements and influence the order in which you want your users to view them. Having a good visual hierarchy can reduce the amount of effort needed by the user to engage with your website or product.

Typography

Typography is the art of arranging letters and text in a way that makes the copy legible, clear, and visually appealing to the user. This includes things like which font you use, sizing, structure, etc.

These are some really simple but really important things to pay attention to if you're doing any kind of design.

JavaScript

So up to this point, we're creating basic websites and making things look nice. Now it's time to get into a programming language and if you're a web developer, you should know Javascript. It may not be the main language that you work with if you're going to be a back-end or even a full-stack dev. You can build websites using server-rendered pages and a template language with PHP or Python, but if you want any kind of dynamic functionality within the browser, then you'll need to know at least some JavaScript because it's the language of the browser.

JavaScript front-end frameworks are extremely popular, but I think in some cases, people move to them too fast. Here are some of the important parts of JavaScript to learn before moving to a framework.

The Basics

The basics of JavaScript are the things that you use in just about every project, so it's important to understand them.

- Data Types
- Data Structures (Arrays, Objects)
- Functions
- Loops
- Conditionals & Operators
- Events
- Browser Console

Many of these things aren't just relative to JavaScrip and are included in just about every programming language. Once you learn them, it makes it easier to learn new programming languages later on.

The DOM (Document Object Model)

In web development, the DOM or document object model is really important. The DOM is an interface in the browser that treats HTML (and XML) documents as a tree structure of nodes. All the HTML tags and text nodes in the document are part of this structure. Nodes can be selected, and manipulated and can have event handlers attached to them. This allows you to have dynamic web pages. When you click a button and something happens on the page, you're working with the DOM.

When you move to a JavaScript framework, a lot of this is done under the hood. They make it much easier to do simple tasks like adding new nodes when an event fires off. With that said, It is important that you learn how to do this with vanilla JavaScript. You don't have to be able to build a Twitter clone with vanilla JS, but at least learn how to create some simple interfaces before moving to a framework.

The browser has a "window" object that has many methods to work with the DOM. You can select any element from the page/DOM and then do things with that element. For example, if you wanted to select a div that had an id of "hello", like this

```
<div id="hello">This is some text</div>
```

You could use the **getElementById()** method on the "document" object

```
document.getElementById('hello')
```

You could also use the **querySelector()** method to select anything, not just IDs

```
document.querySelector('.some-class')
```

To add a new class to that element you could do

```
document.querySelector('.some-class').classList.add('new-class')
```

You can also listen for events lick “click” and perform an action when fired

```
document.querySelector('button').addEventListener('click', function() {
    // Do something when button is clicked
})
```

Asynchronous Programming

JavaScript is a single-threaded language, which means only one thing can happen at a time. This makes asynchronous programming very important. Many times, instead of waiting for a task to finish such as getting data from a server, we want things to keep moving, so we use asynchronous code. Some examples of asynchronous code are the following...

✔ Callbacks

✔ Promises

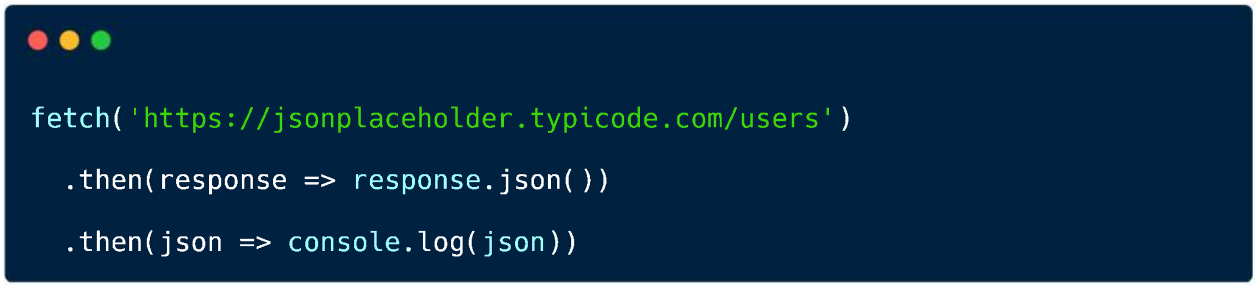
✔ Async/Await

Functions like addEventListener, setTimeout and setInterval have an asynchronous API.

Fetch API, HTTP & JSON

The Fetch API is also something that is important to learn because it allows you to make requests asynchronously to a server to get, post, update or delete data. This is done via **HTTP (HyperText Transfer Protocol)**. As a back-end developer, you need to understand HTTP because you are creating the APIs, but as a front-end developer, you build the part of the app that consumes that API, so everyone should understand HTTP including methods, and status codes, etc. We will talk about REST APIs a little later on.

Let's take a quick look at some code to get some data from an endpoint.



```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(response => response.json())
  .then(json => console.log(json))
```

The code above uses the fetch API to get a list of users from the url https://jsonplaceholder.typicode.com/users . JSON Placeholder is actually a service that provides a fake REST API that gives you data to test with. You can click on the link and you will see the user data. Again, we will talk more about REST APIs and HTTP in the back-end chapter.

When using fetch, it returns a **promise**. We handle that with the .then() method. Fetch actually returns the response itself, then we add a second .then() to get the actual JSON data. In this example, we are just logging it, but you would probably display it in your UI in a real situation.

Most modern APIs return data in a format called **JSON (JavaScript Object Notation)**. It is important to understand how JSON is formatted. It is similar to a JavaScript Object, except both the keys and the string values need to be wrapped in double-quotes. Here is an example of a JSON object.

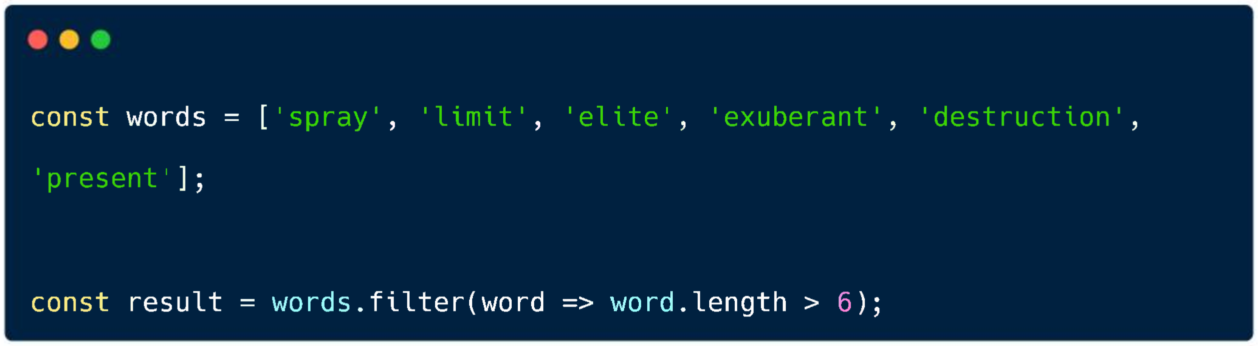
```
{
     "id": 1,
     "name": "Brad Traversy",
     "email": "brad@gmail.com",
     "joinDate": 2022-03-04
}
```

Array Methods

Arrays are a very commonly used data structure in JavaScript and Array methods make using arrays to store and manipulate data very powerful. Each method has a unique function that performs a change or calculation to our arrays so that we don't have to write common functions from scratch. Here are some examples of popular array methods...

forEach()	Loop over an array and execute a callback for each element
map()	Creates a new array with the results of the callback for each element
sort()	Sorts the array elements in asc or desc order
filter()	Creates a new array with the results of the callback and a condition
reduce()	Applies a function against an accumulator and each element in the array to reduce to a single value.
find()	Return value of the first element in an array that passes a test

Here is an example using **filter()** to get all of the words from an array that has a length greater than 6 and put them into a variable.



```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction',
'present'];


const result = words.filter(word => word.length > 6);
```

jQuery?

jQuery is a JavaScript library that is used for many things from DOM manipulation to making AJAX calls and at one point in time (2006 - 2015) it was VERY useful. Manipulating the DOM with vanilla JavaScript was extremely difficult back then and jQuery gave us a whole bunch of helpful methods to create interactive interfaces. AJAX calls before the Fetch API were horrendous. jQuery helped with that, animations and many other things.

In my personal opinion, JavaScript has had many updates since then that have really rendered jQuery useless in many cases.

Today, I don't think that I would suggest learning jQuery. The only reason I may is because you might run into it at some point. There are still many websites that use it. Other than that, there are much better native solutions to most of what jQuery offers. That is my opinion anyway. If you think it would be valuable to learn, by all means, learn jQuery.

Extra Programming Tools

Now let's talk about some other tools that you're going to run into around this time in your learning journey.

Git

Git is one of the tools that **every developer** of any kind should be using. It's an open-source, distributed version control system to track changes in your code. It's used for versioning, backup as well as collaboration. It is extremely helpful when you have multiple people working on a single project. It allows you to see who did what and when they did it.

Your code is stored in a local repository on your machine, but you can push it to a remote repository using a service such as **GitHub**, **GitLab** or **BitBucket**. GitHub is the most popular.

There are other version control systems like Subversion, CVS, etc, but Git is by far the most used and if you become a developer at any company, the chances of them using Git are extremely high.

The basics of Git are actually really easy to learn. There are both command line and GUI clients. I would suggest learning the basic commands. Some of the most used commands are…

git init	Initialize a Git repository
git add	Adds your code to the staging area
git commit -m 'Initial commit'	Commit code to repository
git push	Push to your remote repository
git pull	Pull recent code from repository
git branch new_branch	Create a new branch

These commands are really easy. You will run into issues where you need to roll back your commits, etc. It can be tough in those situations, but you will learn as you go.

Git does the version control, but you also need a remote repository to store your code, *GitHub* is extremely popular, but you also have services like *GitLab* and *BitBucket*.

Markdown

Markdown is a lightweight markup language for creating formatted text. It is something that you'll probably learn along with Git because it is used for Git readme files. Markdown files have a .md extension. It is something that is very easy to learn. You can literally learn it in a day.

Markdown can also be used for books, blogs, etc. We haven't talked about static site generators yet, but it is commonly used for blog posts with a generator such as Gatsby. This makes it so that you don't have to have a database. You can simply create .md files in a directory and they will be displayed on your website.

On the next page is a cheat sheet from https://markdownguide.org. A good place to practice and see your output is https://stackedit.io/app

Element	Markdown Syntax
Heading	# H1 ## H2 ### H3
Bold	**bold text*
Italic	*italicized text*
Blockquote	> blockquote
Ordered List	1. First item 2. Second item 3. Third item
Unordered List	- First item - Second item - Third item
Code	`code`
Horizontal Rule	---
Link	[title](https://www.example.com)
Image	![alt text](image.jpg)

NPM

NPM stands for **Node Package Manager** and it is the official JavaScript package manager. Packages are just code that does a specific thing. It could be a framework, library, plugin, or some type of helper. Just about every language has a package manager, but since JavaScript is what is run in the browser, many front-end developers need to know NPM. It's needed just to set up a front-end framework like React.

As an example, if you wanted to install Axios, which is an HTTP client for making requests to APIs, you would run

```
npm install axios
```

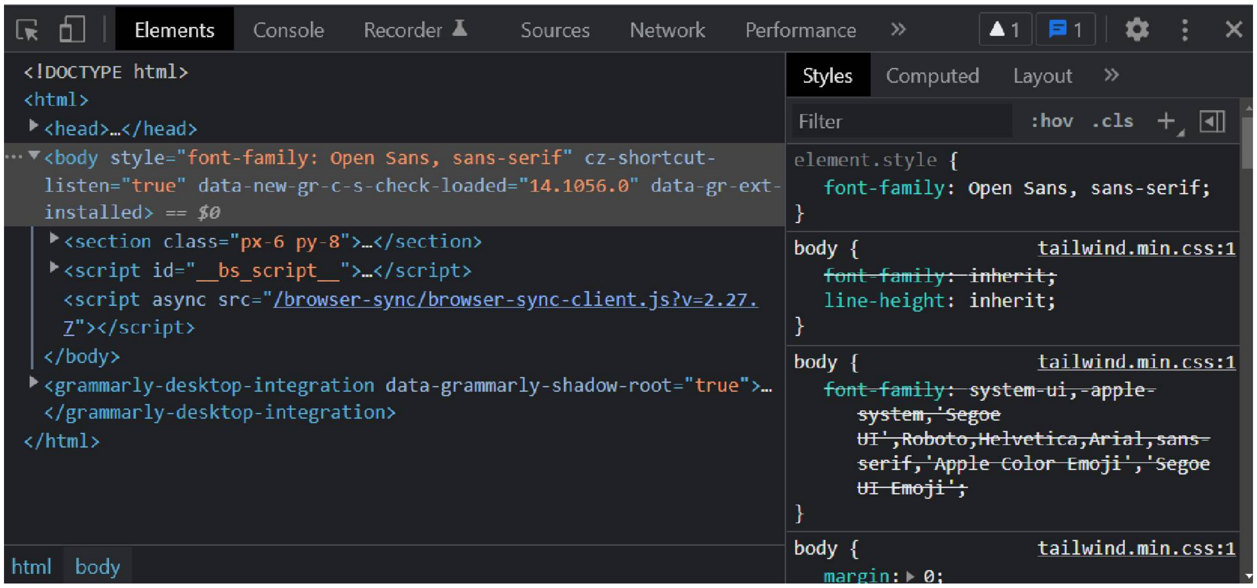
This would then create a directory called **node_modules** and would include Axios and any dependencies that it uses. It would also get added to your **package.json** file which is a manifest that has information about your project such as name, author, and all of the packages/dependencies that it uses.

Yarn

Yarn is another package manager that you can use in lieu of NPM. The commands are a bit different, but they both use the same **npmjs.org** repository. Which package manager you use comes down to preference, like many other things.

Browser Dev Tools

Browser dev tools are extremely valuable when it comes to debugging, viewing the page structure, browser storage, network activity, and so on. All browsers have some type of dev tools that are pretty similar. Try and get familiarized with all of the different tabs and what they do.



```
<!DOCTYPE html>
<html>
 ▸<head>…</head>
…▾<body style="font-family: Open Sans, sans-serif" cz-shortcut-
   listen="true" data-new-gr-c-s-check-loaded="14.1056.0" data-gr-ext-
   installed> == $0
   ▸<section class="px-6 py-8">…</section>
   ▸<script id="__bs_script__">…</script>
    <script async src="/browser-sync/browser-sync-client.js?v=2.27.
    7"></script>
  </body>
 ▸<grammarly-desktop-integration data-grammarly-shadow-root="true">…
  </grammarly-desktop-integration>
</html>
```

```
element.style {
   font-family: Open Sans, sans-serif;
}
body {                                   tailwind.min.css:1
   ~~font-family: inherit;~~
   line-height: inherit;
}
body {                                   tailwind.min.css:1
   ~~font-family: system-ui,-apple-
     system,'Segoe
     UI',Roboto,Helvetica,Arial,sans-
     serif,'Apple Color Emoji','Segoe
     UI Emoji';~~
}
body {                                   tailwind.min.css:1
   margin:▸ 0;
```

There are different tabs for different things. Some of the common tabs you will use are...

- **Elements** - View and edit the pages HTML/CSS
- **Console** - View JS errors/warnings, write JS, etc
- **Network** - View all network and request/response activity
- **Application** - View cookies, localStorage and more
- **Performance** - See how the page performs

There are also many extensions for your browser that can help aid in development. Some of which will add new tabs to your devtools, such as the *React devtools extension*, which will show your entire component tree structure along with any props and state. We'll be going over that stuff soon.

Text Editor Extensions / Plugins

Next, we have text editor or IDE extensions or plugins. Most text editors have the ability to add 3rd party extensions to add to the functionality. Most developers find a suite of tools to install in their editor that helps them code in a more efficient way. So browse around and see what you think will help you.

Emmet is an incredible tool for HTML and CSS where you can use short snippets to write it much faster. It's included with VS Code by default, but you may have to install it as a plugin to some other editors.

Live Server is also great when creating front-end websites and apps. It gives you a development server with live reload.

Prettier and **ESLint** are used for code formatting and linting. There are so many other helpful extensions out there.

When most people start coding, they don't know what is what and it can be very confusing, but after a while, most people curate an environment that works for them with specific browser extensions, editor plugins, themes, settings, etc. VS Code even has a feature where you can sync your settings and environment across multiple machines.

Deploying Front-end Projects

So now that you know HTML, CSS, JavaScript, and some other tools, you'll now need to know how to deploy your projects to the Internet and not just your local machine.

Hosting Platforms

RIght now I am only going to cover some of the platform suggestions that I have for front-end websites and applications. We will talk more about back-end and full-stack deployment and platforms later. They are all companies and services that I have worked with. I don't have any companies sponsoring this eBook.

Netlify

Netlify is one of my favorite services. You can host any modern, front-end project very easily. They do have command-line tools for deployment, however, the easiest and best way to deploy your project is to just push your code to GitHub. You can also use GitLab and BitBucket. Then you can simply log in to your Netlify interface and choose which repository and which branch you want to deploy.

It's simple to set up continuous deployment so all you have to do is push your code to the correct repo/branch and your site gets updated. You also have options for form submissions, serverless functions, environment variables and much more.

As far as pricing, you get a very generous free tier and can host small projects for free. You can visit https://www.netlify.com/pricing for the paid tiers as well.

Vercel

Vercel is very similar to Netlify. The deployment process is the same and they offer most of the same benefits like serverless functions. Next.js is a very popular framework used for full-stack React, which we will talk about later. Vercel is also the creator of Next.js, so they make it very easy to deploy Next.js websites. Vercel also has a very generous free tier. Read more at https://vercel.com.

CPanel Hosting

There are a ton of companies that offer different types of hosting, such as shared, dedicated servers, VPS (Virtual Private Servers). You can get shared hosting for really cheap ($5 - $10 per month). I would probably suggest a VPS even though they are a bit more expensive.

Some examples of these companies are Hostgator, Blue Host, InMotion Hosting, GoDaddy, etc. Most of them use the Cpanel control panel, which lets you manage your hosting account as well as any email accounts that are offered. If you have a dedicated server or VPS with direct access (shell access), you can deploy full-stack applications as well.

As far as deployment and uploading your code, they usually have a few options such as Git, SSH (Secure Shell) or FTP (File Transfer Protocol). FTP tends to be very slow, so I would avoid that if possible.

There are other options for hosting full-stack apps such as AWS, Heroku, etc, but we will talk about those later on.

Domain Names

Domain names are typically going to cost around $10 - $15 per year, so domains are really cheap as long as they aren't already taken and being sold by someone else. I really like Namecheap for domains, but you have a bunch of options such as Domains.com, Bluehost and GoDaddy are still pretty popular for domain names.

Many hosting companies offer a free domain with the purchase of hosting. If not, then you will need to purchase the domain and then point it to your hosting account. Your hosting company docs should have instructions on how to do this. You usually have to add something called "nameservers", so that the domain knows which account to point to.

What You Should Know Up To This Point

Let's do a recap of what you should know up to this point in the guide. You do NOT have to master or be a pro at any of this. You should have some basic knowledge though. Some of what we have talked about is required and some optional.

✓ Create website layouts with HTML/CSS

You should be able to create decent-looking websites with HTML5 and CSS. This includes using semantic tags and styling your sites with CSS including laying everything out with flexbox and/or the grid.

✓ Understand some basic design principles

You are not a designer, however, if you plan on being a front-end developer or a freelancer, you should at least know some basic design principles and know what looks good and what doesn't. As you progress, you will see more layouts and you will get better at this.

✓ JavaScript and how to work with the DOM

You should know JavaScript. The amount of JavaScript that you need to know depends on what your plan is. If you plan on being a Python or PHP developer, you don't have to know as much as a front-end or Node.js developer. Working with the DOM is really important. You should be able to work with events and create simple front-end applications by manipulating the DOM.

✓ Connect to APIs and make requests vis Fetch

You should also be able to work with HTTP by making fetch API requests to some kind of back-end or 3rd party API. Learn about HTTP methods such as GET, POST, PUT and DELETE. We will talk more about this in the back-end section of this book

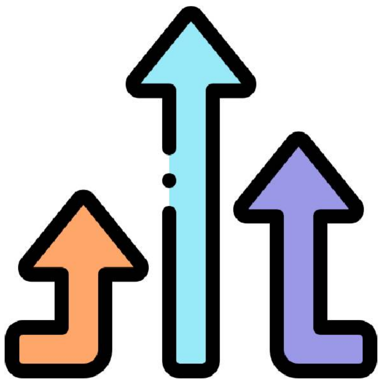
✓ Work with Git

Git is something that every developer should know at least how to work with on a very basic level and push your code to a repository. You could use the terminal or some kind of GUI tool.

✓ Deploy and manage small production projects

We talked about deploying projects. You should be able to create and push a basic website to production and manage the deployment cycle.

As far as getting a job, I think at this point you can start applying for front-end positions, however, these days you will probably have to learn a front-end framework to be a front-end developer. I think that in a lot of cases, depending on the company, you can learn on the job. Having foundational skills in HTML/CSS/JS is really important moving forward and you should have that down.



Where To Go From Here

So now that you know the basics and you can build websites, you have a few choices on where to go from here. Let's explore those choices before moving on.

Path #1 - Web Design

The first option that I have for you is to go the "web designer" route. We don't hear too much about this route in the online web dev community, but it is actually a pretty popular way to go in the real world. This would include getting more into the design, learning more advanced CSS and learning how to create great layouts. You wouldn't really build advanced web applications going this route. It would be more brochure-type websites and "artsy" projects. You would probably be working for yourself providing services for small businesses or working for a small firm.

You may also get into technologies like *WordPress* to give you extra functionality without having to write much code. We'll talk more about WordPress later. You may even use no-code tools such as *Webflow*. I know a lot of freelancers that have successful businesses creating websites and barely write any code. If you are more of a "right-brained" person, this may really be worth looking into.

Path #2 - Front-End Framework

The next route that you could take is to start to learn a front-end framework such as React, Vue or Angular. This would be an option if you plan on being a front-end or a full-stack developer. If you do plan on taking one of these positions, **I would probably recommend that you go this route at this point**. It brings you into a whole new world of technologies including single-page apps, server-rendered apps with frameworks like Next.js as well as static websites with technologies like Gatsby. Every framework has its own ecosystem of certain types of technologies to build different things.

Just make sure that you know a good amount of JavaScript before you start learning any framework. You do not have to be a JS master, but you should know all the basics as well as things like the promises, the fetch API, some array methods, etc.

Path #3 - Back-End

If your plan is to be a back-end developer, then you're probably going to skip the front-end frameworks and start to learn a new language. If you plan on sticking with JavaScript in the back-end, you would start learning Node.js. You may want to use something else like Python, PHP, C#, or Golang, there are so many options. We will talk about all the different languages to choose from later.

You'll also be working with databases like Postgres, MySql, MongoDB, etc. And building JSON APIs and getting more into network aspects and the HTTP request and response cycle.

Path #4 - Advanced JavaScript

Another option is to continue to master JavaScript as a language by doing algorithms and building more advanced vanilla JavaScript applications. You could start getting into some more advanced front-end tools like module bundlers such as *Webpack, Snowpack, Parcel* and *Vite*. These are used to bundle your JavaScript and other assets, and create modules and a much more intricate file structure. Many frameworks use these under the hood, but you can also use them for larger vanilla JavaScript applications or even to build your own framework, which is something that I've always wanted to do.

You could also get into testing and things like software design patterns. I think for a hobby, this would be a good choice, but if you are looking to build more practical applications and look for a job, it may not be the best thing to do at this point.

CHAPTER 3

FRONT-END ADVANCED

In this chapter we will talk about more advanced tools that you will need to learn as a front-end or full-stack developer, such as frameworks, testing, server-rendered pages and statically generated websites.

Front-end Frameworks

Like I stated in the last chapter, I think that this is the best option to move to next for front-end developers as well as full-stack devs that want to create really interactive web applications. You can create server-rendered pages using template engines with PHP, Python and other languages, so it is not mandatory for full-stack developers, but it will help you create better UIs for your apps and will help you get jobs.

What These Frameworks Have In Common

No matter what framework you choose, you will find that they all kind of do the same thing, just in different ways. The following concepts are common, no matter which framework you want to learn.

Single Page Applications

When I say “front-end”, that refers to the client-side and what are called **Single Page Applications** or **SPAs**. This means the browser only loads a single HTML page from the server and then everything else is done through JavaScript using the framework. React, Angular and Vue all have a **single index.html file** and the only actual HTML that is in the body is something like this…

```html
<div id="root"></div>
```

Then it loads a JavaScript bundle where it takes this “root” element and inserts the entire frontend with all of the components and output.

When data is loaded, it is requested from a server/backend via AJAX calls using something like the Fetch API or Axios and then loaded into the UI.

On the next page is a diagram to help visualize this.

TRADITIONAL WEB APPLICATIONS

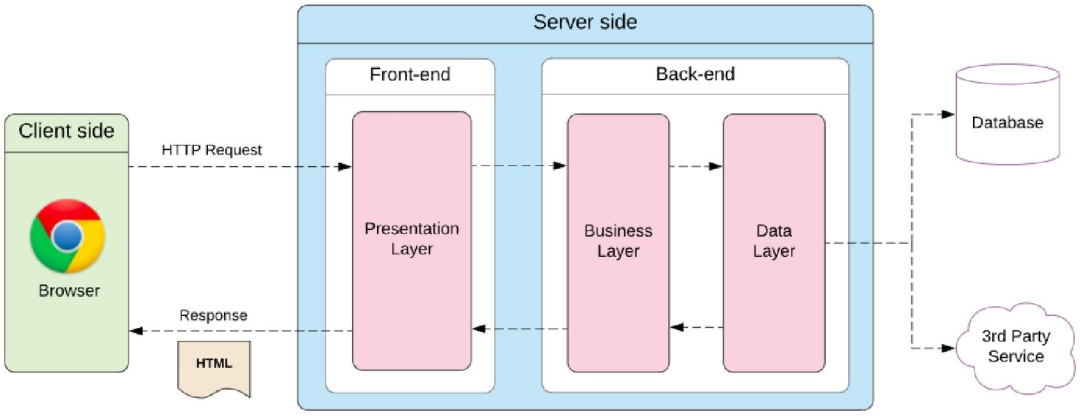


Image from elmprogramming.com

SINGLE PAGE WEB APPLICATIONS

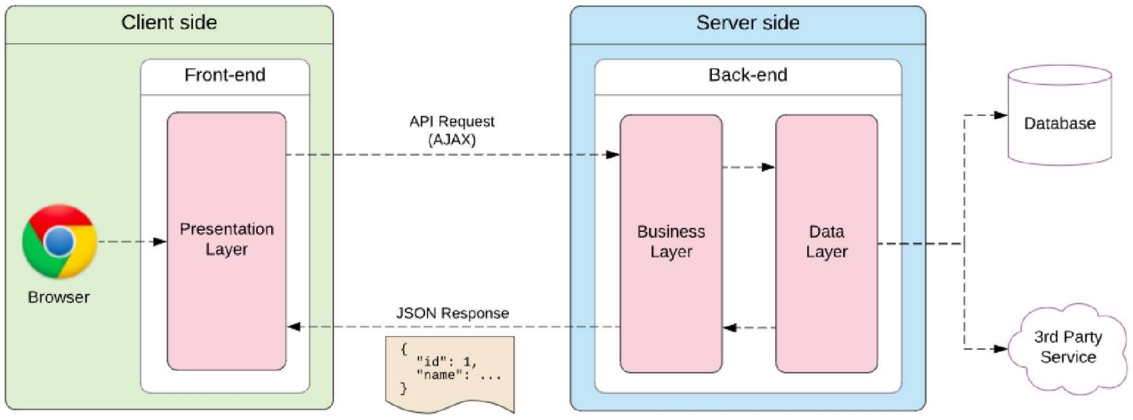


Image from elmprogramming.com

There are ways to run these frameworks on the server-side, which we'll talk about soon, but these are inherently client-side or front-end frameworks.

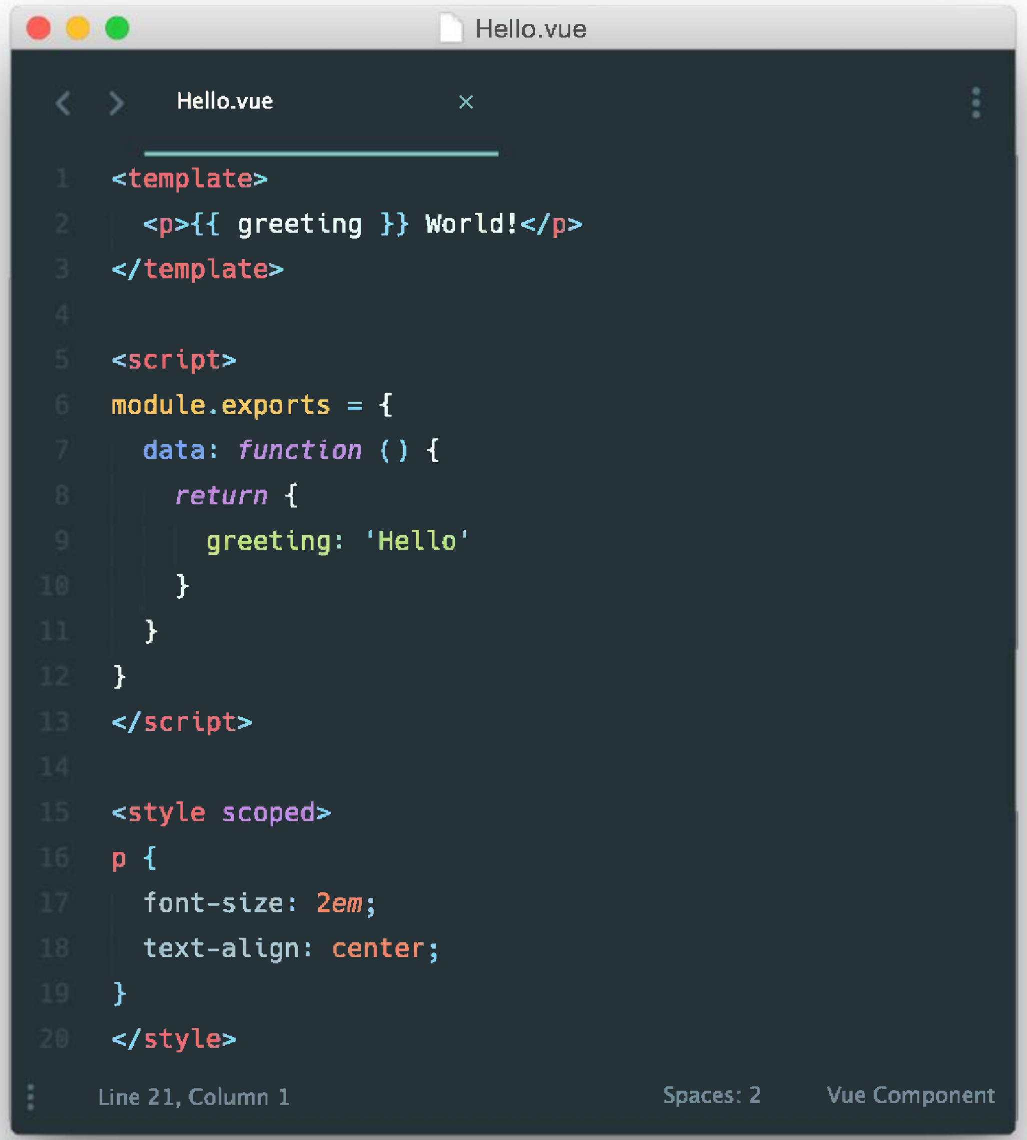
Components

There are a lot of similarities with these frameworks. They all use something called “components”, which are basically pieces of the UI. So menus, toolbars, navigation, forms, etc are all created as components that encapsulate the output (HTML), the logic (JavaScript) and sometimes the styling (CSS).

Some components may simply be HTML output that don’t do anything else and some may be “smart components” that have logic as well.

Every framework has a different way of creating a component. React used to mainly use classes, because they could hold what we call “state”. React now uses something called **hooks**, that allow us to use state within function-based components. So that is typically what people use now.

On the next page is an example of a very simple Vue.js component



```
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
module.exports = {
  data: function () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>

<style scoped>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

It has the output/HTML at the top, then the logic (any state, methods, etc) and then the styling at the bottom. So everything gets encapsulated.

State

You will also be working with state when dealing with a front-end framework. State is basically just data. There is **local component state**. This would be something like a boolean value (true/false) in a menu component pertaining to whether it should be open or closed. Local state only has to do with a single component's properties.

Then we have **global** or **app-level state**, which is data that is needed across multiple components. If you have a blog, the blog posts that you fetch from the back-end will probably be put into global state so that any component can access them. There are many ways to handle state including built-in methods as well as 3rd-party state managers like Redux.

Now let's look at some of the actual frameworks that are available to us. Remember, these are "Front-End" frameworks. Something like Express, Laravel, Django, etc are "Server-Side". We will talk about those later on.

React

First we have React, which is the most popular framework by many metrics. It's technically a UI library because it doesn't have everything that a "framework" usually would have included in the core package. For instance, it doesn't have a router built in. However, there are commonly used packages like *react-router-dom* that you would simply install with NPM or Yarn and then you have that functionality. **It is also in direct competition with frameworks like Angular.**

React was created and is maintained by Facebook. As far as the learning curve, I would say it's moderate. It's not the easiest and not the hardest.

Things you will learn along with React:

■ **Create React App:** A CLI tool to generate React apps and run a dev server

■ **React Router:** A package to create routes and link to those routes in React

■ **JSX (JavaScript Syntax Extension):** The output of a React component. It is basically HTML within JS

■ **Hooks:** Used in functional components to hook into the lifecycle and access state

■ **Context API:** A built-in API to manage global state within React

■ **Redux:** A 3rd party global state manager. Good for large apps with a lot of state.

A lot of people ask me if I think we still need Redux now that we have the context API and hooks. I would say that **Redux is still superior for really large applications**, but that is just my personal opinion.

Vue

I would say that Vue.js is the second most popular framework. It doesn't have the backing of a huge company and it still made it to where it is, so that tells you that it's pretty damn good. There's a great community around it and it's fairly easy to learn when you compare it to React or Angular.

I love the layout of components in Vue. As you saw in the image above, you literally have a file with the template/HTML, the logic and the styling for that component. This makes it really easy to work with.

Things you will learn along with Vue:

■ **Vue CLI:** Command-line interface to scaffold and manage your Vue projects

■ **Vue Router** - Add routing and links to Vue apps

■ **Vuex** - A state manager for Vue

■ **Composition API** - A new way to write Vue components with version 3

Angular

Angular was created by Google and is still pretty popular in the enterprise world, but not so much around startups and smaller companies. I would definitely say that it has the steepest learning curve out of the four, but it is a much larger framework that includes a lot in the core packages. It has its own router, its own HTTP client, and so on. It also uses TypeScript by default. I will talk about TypeScript soon as well.

Things you will learn along with Angular:

■ **Angular CLI:** Command-line interface to generate and manage Angular projects

■ **HTTP Client:** Module for making HTTP requests

■ **Services:** Used to share data across components

■ **Observables:** A way of working with asynchronous code

■ **TypeScript:** A superset of JavaScript that offers static-typing

Svelte

Svelte is gaining a ton of traction. It is not technically a framework but does a lot of the same thing. It's actually a JavaScript compiler. So you build your app in a certain way using .svelte files and it runs through a compiler and compiles to pure JavaScript. No framework code whatsoever. So it's very lightweight and performant.

I would definitely say that Svelte has the easiest learning curve and in all honesty, sometimes frameworks like React and Angular seek to make certain things overcomplicated, where Svelte feels very natural and just makes things work as you would expect.

Some things you will learn with Svelte:

■ **SvelteKit:** Includes all you need to build powerful Svelte apps including routing, server-side rendering, etc

Browser & Text Editor Extensions

Just about every framework has a browser extension to help aid you in development. Browser extensions will show you your component tree and state values, which is extremely helpful.

You should also look for any popular text editor/IDE extensions for things like highlighting, autocomplete, etc.

Choosing a Framework

Choosing a framework is completely up to you. Don't listen to anyone that says "X is the best framework". A lot of it is preference. Of course, popularity matters if you're looking to get a job. So look at listings in your area and see what's being used. My advice would be to **try all of them and choose one that you feel clicks with you.** That's one of the reasons that I create crash courses. So that you can build the same project with all of them and see what you like the best.

TypeScript

The next technology that we're going to talk about is TypeScript, which is getting a ton of traction. I believe it was StackOverflow's second most loved language in 2021.

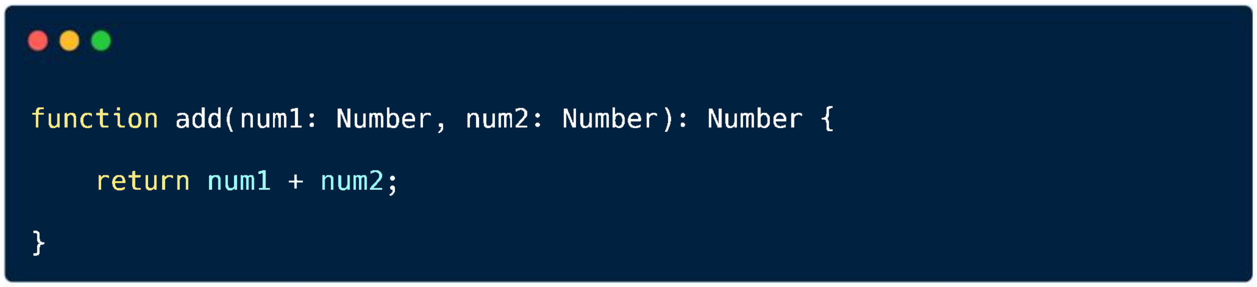
TypeScript is a *superset of JavaScript*. That means that it includes everything that JavaScript does so it is essentially JavaScript, but it has some additional features. The main feature and what most people use it for is **static-type checking**, hence the name "*TypeScript*".

Dynamically-Typed vs. Statically-Typed Languages

Javascript is a *dynamically-typed* language meaning that when you define a variable or a function, you don't have to define the type of that variable or what the function returns. When I say *"type"*, I mean string, number, boolean, etc. Instead of you having to assign the type, it is dynamically interpreted for you. This is the same with languages like Python, Ruby and PHP.

Some languages like C, C++, Java, etc are *statically-typed*, which means that you do define the types manually. Whatever you define, that value has to be assigned and stick to that specific type. Yes, it is more code to write, but there are benefits to doing this. It makes your code less prone to errors, more robust and more declarative.

Here is an example of a very simple TypeScript Function...

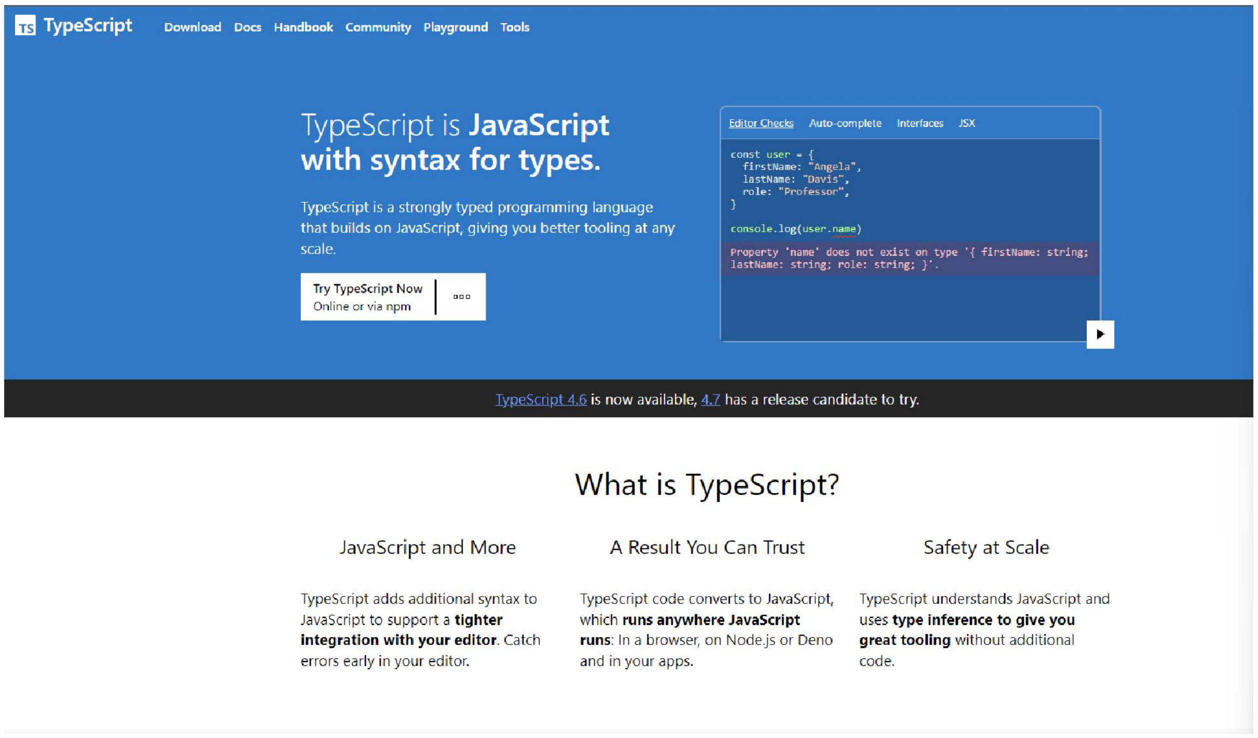


```
function add(num1: Number, num2: Number): Number {
    return num1 + num2;
}
```

As you can see, TypeScript adds the option to add types to JavaScript. It is completely optional. TypeScript also offers class and module support and ES6+ features like arrow functions.

You can use TypeScript on its own with the TypeScript Compiler (TSC), but it is also easily added with tools like Create React App, Next.js, etc.

As far as learning TypeScript at this point, I think it's completely optional, but it does have its benefits. It's used a lot at companies and it's used with front-end frameworks as well as on the back-end with **Node.js**. I would suggest trying it out and then weighing the pros and cons and coming to your own conclusion.



TypeScript is **JavaScript with syntax for types.**

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.

```
const user = {
  firstName: "Angela",
  lastName: "Davis",
  role: "Professor",
}

console.log(user.name)
```

```
Property 'name' does not exist on type '{ firstName: string;
lastName: string; role: string; }'.
```

TypeScript 4.6 is now available, 4.7 has a release candidate to try.

What is TypeScript?

JavaScript and More

TypeScript adds additional syntax to JavaScript to support a **tighter integration with your editor**. Catch errors early in your editor.

A Result You Can Trust

TypeScript code converts to JavaScript, which **runs anywhere JavaScript runs**: In a browser, on Node.js or Deno and in your apps.

Safety at Scale

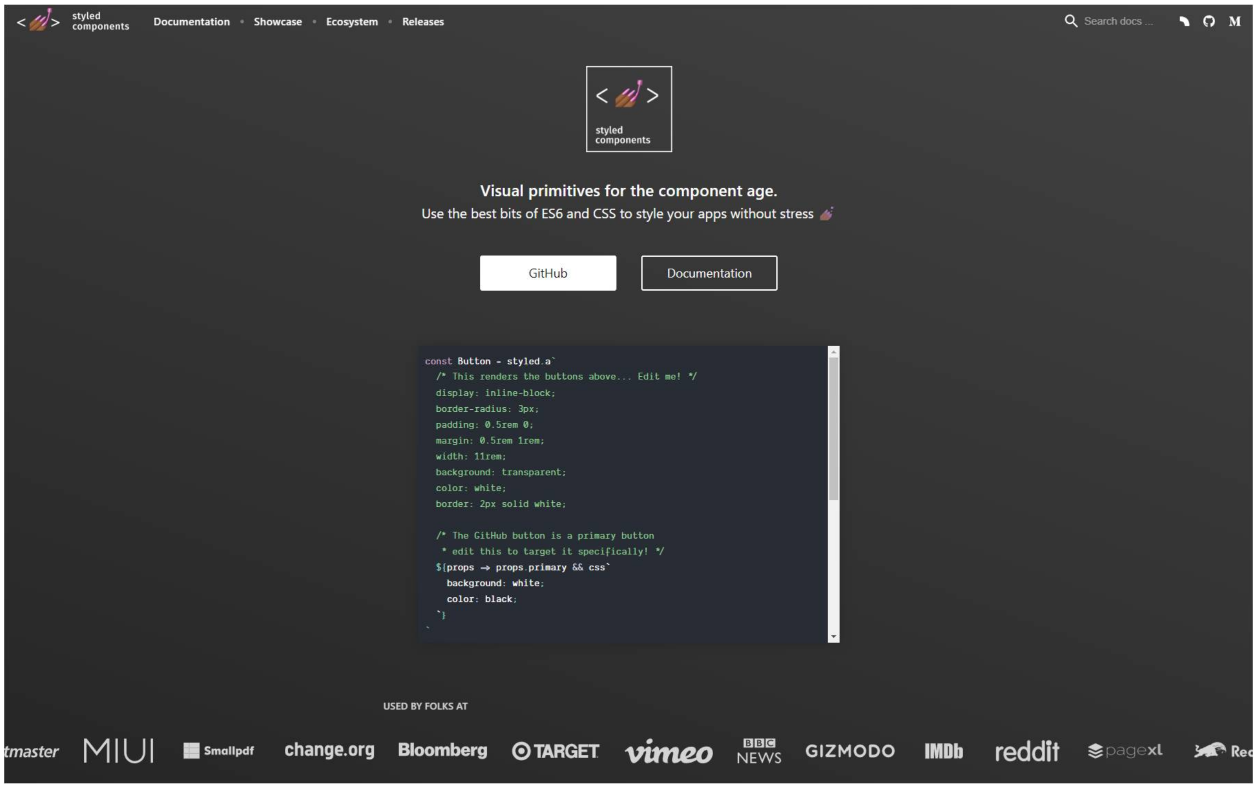
TypeScript understands JavaScript and uses **type inference to give you great tooling** without additional code.

Visit https://www.typescriptlang.org for more info

UI Kits/Libraries

Now that you know a front-end framework, you have a lot of choices for styling and design. Of course, you could just use a global stylesheet or you could have separate stylesheets for each component. You have some other options as well.

Styled Components is a great library that allows you to create just that, components with the purpose of styling. So you may have something like a "*Card*" component that you would wrap around content that you want in a card.



Visual primitives for the component age.
Use the best bits of ES6 and CSS to style your apps without stress

```
const Button = styled.a`
  /* This renders the buttons above... Edit me! */
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 11rem;
  background: transparent;
  color: white;
  border: 2px solid white;

  /* The GitHub button is a primary button
   * edit this to target it specifically! */
  ${props => props.primary && css`
    background: white;
    color: black;
  `}
`
```

https://styled-components.com

You also have the option of using a UI kit or sometimes called a UI library. These come with pre-made components for things like navigation, modals, text styling and many other things.

Each framework has a bunch of UI kits that you can choose from, so let's look at some of the popular ones.

React UI Kits

Material UI

Material UI is probably the most popular kit for React. It is based on Google's Material design. This is my pick, but it's really going to be your preference, especially where it has a lot to do with design and aesthetics.

Chakra UI

Chakra UI is a simple, modular and accessible component library and I think is a bit easier to use than Material UI. There seems to be less code to write to get the same type of effect.

Onsen UI

Onsen UI is a beautiful and efficient way to develop HTML5 hybrid and mobile web apps and is available for React as well as Vue and Angular.

React Bootstrap

React Bootstrap, as it implies, is a UI library based on the Bootstrap CSS framework. So all of the elements in Bootstrap are available as React components.

Vue UI Kits

Vuetify

Vuetify is definitely my favorite kit for Vue. It has a ton of really professional-looking interactive UI elements and I would say is by far the most popular for Vue. It's also based on Material Design.

Vue Material

Vue Material is another Material Design-based UI kit. It's fairly lightweight and it isn't as opinionated as many other libraries.

Buefy

Buefy is also pretty popular. It's based on the Bulma CSS framework. What I like about this is that it only requires minimal code to use the elements.

Bootstrap Vue

This is of course another Bootstrap-based UI kit. It's pretty widely used with over 11,000 GitHub stars. If you know Bootstrap, you'll pick this up pretty easily.

Angular UI Kits

Angular Material

Angular Material is a Material Design kit. It's one of the most popular kits for Angular. It has a good number of components and good accessibility features.

Ng-Bootstrap

This library is based on Bootstrap. Angular actually has quite a few Bootstrap kits. I think this is the most popular. It's easy to use and customize.

Ng-Zorro

This library is an enterprise-class kit and is based on the Ant design. It's well structured with pretty good documentation.

MD Bootstrap

MD Bootstrap is one of the biggest UI libraries with tons of components. There is a pro version available as well.

Svelte UI Kits

Svelte Material UI

Svelte Material UI is a library of Svelte 3 components with touch/mobile-optimized design. It's based on Material Design.

Smelte

Smelte is one that I haven't used yet but looks interesting. It is based on a combination of Svelte and Tailwind CSS.

Sveltestrap

And of course, we have to have a UI library for Svelte that is based on Bootstrap.

Alpine.js

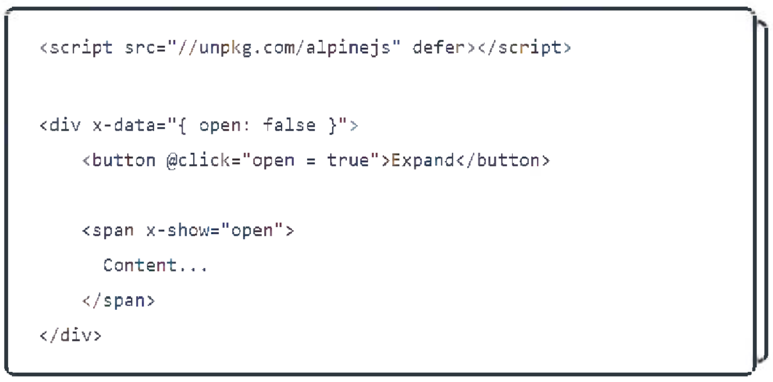
I wasn't really sure where to fit this in, but I really want to talk about Alpine.js. It is technically a front-end JavaScript framework, however, it is much more lightweight and used in a different way than the others.

When you use React or Vue, you build your entire app around that framework's structure. Alpine is more like JQuery, where you simply include a CDN and you use it where and when you want. It could be in a static HTML/CSS website or a Laravel or Django application that uses templates.

Everything is done within the HTML. You have custom attributes like *x-data* and *x-show* and the values of those attributes can be JavaScript code. This makes it great for websites where you want to add a slider, carousel or some kind of interactive component without having to create a React or Vue app.

Alpine.js

Your new, lightweight, JavaScript framework.



```
<script src="//unpkg.com/alpinejs" defer></script>

<div x-data="{ open: false }">
    <button @click="open = true">Expand</button>

    <span x-show="open">
      Content...
    </span>
</div>
```

As you can see from the image above, it is simple to create a button to toggle something on the page. We don’t have to leave the HTML at all.

For more info visit https://alpinejs.dev

Testing

Testing is something that I have often said I don’t do enough of. I seem to hear this from a lot of developers. I think that some people feel that the trouble and time that it takes to test your applications isn’t worth it. I do see that argument. However**, once you take the time to learn how to test, I think you’ll see the benefit** and that it is worth it for a lot of your projects. I think it’s especially useful in large projects.

There are different types of testing and every language has tools for testing. I’m going to talk about 3 common testing types, but there are more than this. You have sanity testing, regression testing, etc, but what I would suggest is starting with the most common, which are these unit testing, end to end testing and integration testing.

Unit Testing

Unit testing is where you test specific blocks of code, usually functions, modules, classes, etc. It's so you can make sure that the individual parts of your code work properly. Developers often execute unit tests through test automation.

By writing tests for the smallest testable units, and the compound behaviors between those units, you can build up comprehensive tests for complex applications.

Integration Testing

Integration testing is usually done in concert with unit testing. Unit testing tests blocks of code and integration testing will then test how those modules or blocks of code work together. The purpose of this type of testing is to expose defects between modules when integrated

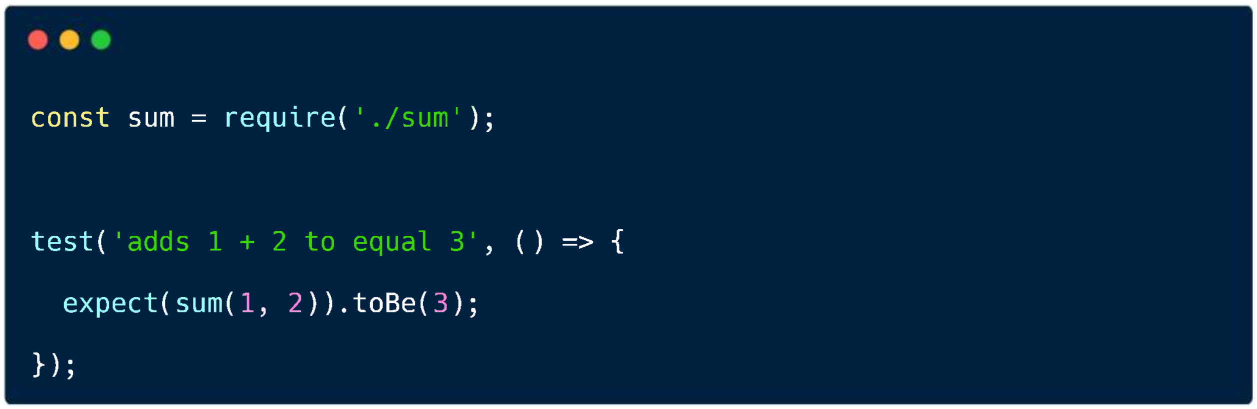
End To End Testing

E2E testing does just that, it tests from beginning to end to ensure the application flow behaves as expected and we usually do this by emulating a user and going through real user scenarios. This is sometimes called system testing.

As far as tools, every language has testing frameworks and libraries. Here are some of the popular ones for different languages.

- **JavaScript:** Jest, Cypress, React Testing LIbrary.
- **Python:** PyTest, PyUnit.
- **PHP:** PHPUnit, Storyplayer
- **C#:** MSTest/Visual Studio
- **Java:** Selenium, Robot Framework
- **Go:** std Library testing package, Testify
- **Rust:** Skeptic, rustdoc
- **Ruby:** Cucumber, Watir

Here is a very simple example of testing a sum() function with Jest



```
const sum = require('./sum');


test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Web APIs

There are a bunch of HTML5 and JavaScript APIs that are available in the browser environment. I think at this point, you should know some of these and they can be very useful for building applications. We will discuss some of these, but you can see a full list here.

GeoLocation API

The GeoLocation API provides a way for users of your application to provide their location. I'm sure all of you have had some kind of website ask if you want to share your location. This is GeoLocation. The API is pretty simple to work with. There are a couple methods to use.

■ **Geolocation.getCurrentPosition()** - Gets the devices location
■ **Geolocation.watchPosition()** - Registers a handler function that will be called when the position changes

This API is extremely useful for any application that needs to fetch or watch a user's location. For instance, if your app shows the closest restaurants.

Audio & Video APIs

Back in the HTML/CSS Section, we looked at a simple example of the <video> and <audio> tag. With those HTML5 tags, comes a JavaScript API. This can be used to make your own more advanced video and audio players.

When you select a video/audio element in the JS, you can use the following methods.

- **play()** - Play the video/audio
- **pause()** - Pauses current video/audio
- **addTextTrack()** - Adds new text track
- **canPlayType()** - Checks if browser can play specified file
- **load()** - Reloads the audio/video element

There are also a ton of properties. Here are some of them

- **currentTime** - Returns the current playback time
- **duration** - Gets the duration of the item
- **readyState** - Returns the current readyState
- **loop** - Sets or returns the media to start over when finished
- **volume** - Sets or returns the volume

Drag & Drop API

Drag and drop uses the DOM event model and drag events inherited from mouse events.

You can respond to the following events in your JavaScript code.

Event	Handler	Fires when...
drag	**ondrag**	A dragged item (element or text selection) is dragged.
dragend	**ondragend**	A drag operation ends (such as releasing a mouse button or hitting the Esc key; see Finishing a Drag.)
dragenter	**ondragenter**	A dragged item enters a valid drop target. (See Specifying Drop Targets.)
dragleave	**ondragleave**	A dragged item leaves a valid drop target.
dragover	**ondragover**	A dragged item is being dragged over a valid drop target, every few hundred milliseconds.
dragstart	**ondragstart**	The user starts dragging an item. (See Starting a Drag Operation.)
drop	**ondrop**	An item is dropped on a valid drop target. (See Performing a Drop.)

Canvas API

Canvas is used for drawing items using JavaScript. You use the <canvas> element and create a 2D context to create lines and shapes.

```html
<canvas id="canvas"></canvas>
```

Then in the JavaScript…

```javascript
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');

ctx.fillStyle = 'green';
ctx.fillRect(10, 10, 150, 100);
```

The result would be

Obviously this is a very simple example, but you can create some pretty advanced 2D games with the canvas api.

Web Speech API

The web speech API is great for speech recognition as well as generating voices and text to speech.

The **SpeechRecognition()** interface is the controller for the recognition service. There are methods to start and stop listening for incoming speech.

The **SpeechSynthesis()** interface can be used to retrieve info about the synthesis voice available on the device and start/stop speech.

Web RTC

Web RTC is a very powerful API used for real-time communication. You can build chat apps and optionally stream both video and/or audio. There are several interrelated APIs and protocols which work together to achieve this.

Dennis Ivy did a great 4 hour project building a video chat app using Web RTC.

The Websockets API

Websockets allow you to create a two-way session between the client and server. Typically the client sends a request to the server and gets a response. Think of it as sending a messenger from one house to the other. With websockets, it is more like two neighbors across the hall communicating with the doors open.

This makes for great real-time applications such as chat apps or any kind of CRUD (create, read, update, delete) app.

Socket.io is a popular event driven library that utilizes websockets to create real-time applications. This is more of a full-stack technology, because you will need a server.

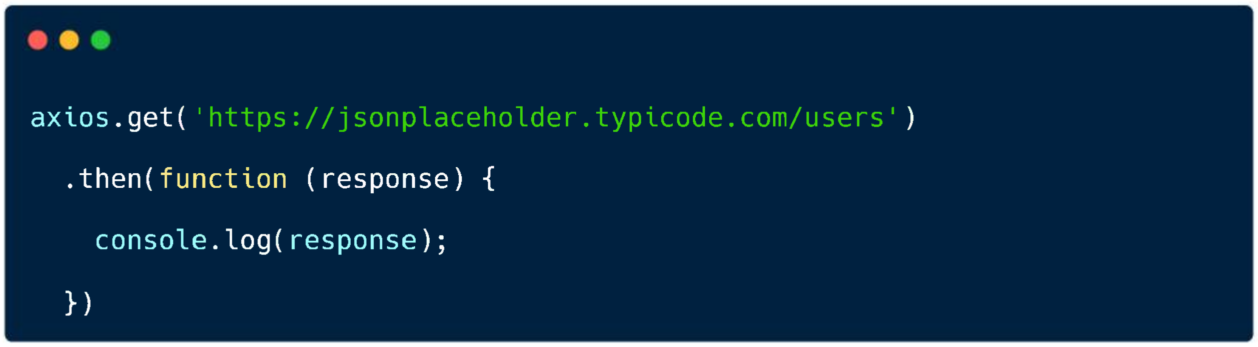
To read more about Socket.io, visit https://socket.io

There are a lot more web APIs available. I just can't talk about all of them. Check them out here.

Axios

We talked about the native fetch API a while ago. That is one way to make an HTTP request from your front-end, but there are also 3rd party libraries to do that as well and one that is really popular is Axios. To use Axios, you can include the CDN or you can install it with NPM.

The syntax to make a request is a little cleaner than with Fetch. In our Fetch example, we had to use two .then() methods on the returned promise. With Axios, we only need one. This is what it would look like.



```
axios.get('https://jsonplaceholder.typicode.com/users')
  .then(function (response) {
    console.log(response);
  })
```

As you can see, we only need one .then() and that gives us the response with the JSON data.

Server-Side Rendering

This is probably my favorite thing about modern web development over the past couple of years and that is getting away from doing everything on the client and moving to render pages on the server while still using a framework like React or Vue.

This is where the whole front-end and back-end line gets to be really blurry. You're using a front-end framework, but it's no longer a straight single-page application running on the client.

All four frameworks that we talked about have what I like to call "parent frameworks" that we can use to render pages on the server. I know that sounds strange and I know that a lot of people's first thought is "great, another framework". However, I would suggest thinking of them as building React, Vue, Angular and Svelte apps in an environment with some extra benefits that make your life easier and to make your users happier. It's not like learning a whole new FE framework. If you know React, then learning Next is not bad, because for the most part, it IS React. You just have some extra features.

One huge advantage to using PHP, is that you could just create a **.php** file and upload it to your server and then go to that page in the browser. Well with these SSR frameworks, we can do the same thing and create a page like "about.js" and then upload it to the "pages" folder and it just works.

I will talk more about the benefits of SSR below, but first, let's look at some of our options.

Next.js

Next.js would be my pick but it is based on **React**, so if you're a Vue developer, you probably would use Nuxt instead, which I'll get to in a minute. Next.js has been around for a while and is just fantastic. You can render your pages on the server and have access to data fetching methods where you can run server-side tasks and have access to the HTTP request data before the page loads.

You also have a folder for API routes. So you can have your API/backend in the same folder structure as your front-end stuff as opposed to having it be something completely separated.

You also have extra tools such as image optimization and static site generation, which I will talk about soon.

Nuxt.js

Nuxt.js is very similar to Next except instead of React you build your site with Vue.js. So naturally, if you're a Vue developer, you'll most likely use Nuxt. It offers a lot of the same features, so I won't repeat everything that I just stated about Next.js.

Remix

Remix is very new and is another option for React developers. I've been getting into it recently and I love it. I just did a 2 part crash course on it. Although it shares some features with Next.js, it feels very different. Remix has something called "loaders" that allow you to load data from the server and "actions" in which you can submit data to the server. I think that these are the best features of Remix and make it really powerful and give you the most control over the HTTP request/response cycle.

SvelteKit

SvelteKit is your option for Svelte. I honestly would suggest using SvelteKit for just about any Svelte app. It offers server-side rendering, advanced routing, code splitting, and much more.

Angular Universal

Angular Universal is your option for SSR with Angular. Instead of executing in the browser like a normal Angular app, Angular Universal executes on the server, generating static application pages that later get bootstrapped on the client. The app then renders more quickly.

Benefits of SSR

Let's look at some of the reasons why you would want to use server-side rendering over single-page applications.

✓ SEO

Search engine optimization is a huge one. When you build a single-page app (SPA), the browser loads a page, and then it loads a JavaScript bundle that includes your entire front-end. The problem with that is the search engines can't crawl your pages because they're loaded afterward through JavaScript. This has been an issue with SPAs for a long time. Now that we can render pages server-side, they get loaded as a regular HTML page that the search engines can crawl.

✓ File-Based Routing

Routing is much simpler with server-side rendering. Instead of having to define all of your routes you simply put specific files/components in a specific folder and the routes just work.

✓ Performance

You also get performance benefits by being able to load pages faster that are already pre-filled with your data. As opposed to waiting for the client-side JavaScript. This also makes for a better user experience.

✓ Convenience

It's convenient to have everything in one place. For instance, with Next.js you can create API routes in the same folder structure and so server-side tasks rather than always having a completely separate back-end.

✔ **Improved Data Security**

You also get some improved data security because obviously data is more secure on the server-side than on the client where everyone has access to it.

I honestly think that this is going to be the new norm for front-end frameworks, or at least for larger projects.

Static Site Generators

The next batch of technologies that we have are static site generators. These allow you to generate your websites and content using data from various sources such as markdown files, some kind of API, or maybe a headless CMS, which I'll talk about next. These frameworks get all data ahead of time and then preload it so when the user comes to the page everything loads very fast. As opposed to coming to the page and waiting for the data to be fetched.

Now there are some drawbacks to this because if you have data that is constantly changing, you probably don't want to do things in this way, but if you have a blog or something where you update the data once per day or a few times per week, static sites are a great option.

What’s great is that we have some static site generators already built into some of the front-end framework ecosystems.

Gatsby

Gatsby uses the React framework. You create pages that are simply React components. So you can still structure everything as you would in a regular React app, but with extra capabilities. Gatsby has a Graphql data layer to work with as well. I haven’t gone over Graphql yet, but I will in a bit. Gatsby has all kinds of features such as image optimization, code splitting, and more.

Next.js

As we just talked about, Next.js renders React on the server, however, it can also be used as a static site generator. There are some specific rules to follow in your code, but Next.js has an “export” command to export your project as a static site. It has data fetching functions for fetching whatever you want as well as creating a dynamic URL structure for your static files.

Gridsome

Gridsome is very similar to Gatsby but instead of React, it uses Vue.js. So naturally, if you're a Vue developer, you would probably go with Gridsome over Gatsby. It has a bunch of features for PWAs, SEO, and more.

Eleventy (11ty)

Eleventy is a simple to use and powerful static site generator that is capable of mixing template languages. Content can be written with Markdown or one of many other template languages.

Jekyll

Jekyll is a static site generator that has been around for a long time. It isn't associated with any JavaScript framework. It's actually Ruby-based and it uses the Liquid template engine.

Hugo

Hugo prides itself on being the *"fastest framework for building websites"*. It is a static site generator based on the *GO* programming language. It’s fast and flexible and you are not bound to any specific front-end framework.

Benefits Of SSG

✓ Speed & Performance

When you generate a site with something like Gatsby, All of the HTML files are being produced directly for you to upload to the server. They’re just static files. So there isn’t any need to query a database or anything. Everything has already been generated. So static sites are super fast.

✓ SEO

SEO is another big one because your pages are nothing but straight HTML pages, which is what the search engines love.

✓ Simplicity

Simplicity is another benefit. There aren’t any databases, packages, libraries, etc. They’re much easier to manage than a dynamic site. Of course, you do have some limitations as to what static sites can offer.

✓ Security Benefits

Since static sites have no database, you can't have SQL injections, etc. So you get those security benefits.

✓ Easy Deployment

Static sites can be deployed virtually anywhere because they're just static files. So you can even deploy right to a CDN.

Headless CMS

So when we think of a content management system (CMS), a lot of us, especially if you've been doing this for a while, think of WordPress or Drupal or if you're really old like me, you may even remember Joomla, PHPNuke, etc. These are systems that you and your clients can log into some kind of admin area and add and manage content. Then it would reflect in the front-end where you had some kind of theme.

A headless CMS has that admin area functionality, but there is no front-end. You're free to **use whatever you like on the front-end** because the content that you create simply spits out a JSON API. So you can then use React or Svelte or no framework at all and use your CMS data to create pages, menus, etc. There are also authentication systems built in that usually have to do with tokens. You can also use tools like Auth0. I wasn't sure where to actually fit these in because technically they are "back-ends", however, they're easy enough for front-end developers to use because you're not really coding your back-end from scratch. So it’s kind of a full-stack technology. The lines are really blurring between what’s considered front-end and back-end.

There are a lot of headless CMS out there. These are some of the most popular and some of my favorites.

Strapi

Strapi is currently my personal favorite. It is completely open-source and self-hosted, meaning that you set it up yourself on your own server. Unlike some other systems where your back-end is hosted on their servers. You can create content types like you would in WordPress. There’s a plugin system to expand functionality. Overall, it’s powerful and easy to use.

Sanity.io

Sanity.io is an incredibly popular headless CMS. It’s a powerful and flexible platform. It’s free to use up to a certain point, but there are some pricing tiers. Looking at the features of Sanity, it seems like a great choice for collaboration. The way content is structured makes it easy to have multiple people working on a project. There's also a toolkit called Sanity Studio that you can use as well.

Contentful

Contentful is a headless CMS that seems to be really popular in the big business world, It has features that are great for teams. It's really fast and optimized for speed. They have features like an images API that can retrieve and manipulate images among other unique features.

Ghost

Ghost is a headless CMS and is also marketed as a blogging platform. In some ways, it directly competes with WordPress. You can install it on your own server and you can also use the hosted solution, much like WordPress.

GraphCMS

GraphCMS is something that I recently discovered and it allows you to instantly create a GraphQL API. We're going to talk about GraphQl soon.

Some of these other content management systems like Strapi also allow you to use GraphQL, but Graph CMS really focuses on it.

WordPress

Yes, **WordPress can also be used as a headless CMS**. You can use the admin area and instead of having a template and generated HTML/CSS, you simply get a JSON API. That way you can have whatever you want in your front-end. You could use React, Vue or anything else.

I think headless content management systems have a bright future in web development because they take so much work out of creating a backend. From database integration to creating routes manually, middleware, authentication and so much more that you get out of the box for free.

The Jamstack

So the last few technologies that we looked at can fit into something called the *Jamstack*. This is a very broad term for an architectural approach to building websites. It was originally presented with a capital JAM which stood for **JavScript, APIs, and Markup**. From what I understand, they got rid of that acronym as it's defined more broadly. You can visit https://jamstack.wtf to really see what it's all about, but here is some summarized information about the Jamstack.

Variety of Tools / Technologies

Jamstack sites can use a huge variety of tools and technologies. An example would be a blog website with a static site generator like Gatsby along with a headless CMS for data. Let's say we have all of our data coming from Sanity.io. We have Gatsby to get the data from Sanity and then spit out static files with our blog posts. So we deploy the static site and it's extremely fast because there's no fetching from a database or anything. That was done at build time and then created static files for us. This would be an example of a Jamstack project

Decoupled

Everything is decoupled. **The front-end tooling is separate from the back-end**. As I said, the front-end is typically built using a static site generator. You can also use *serverless functions* to perform server-based operations without having an actual server that you maintain yourself. Platforms like Vercel and Netlify allow you to run functions to do server-side tasks.

Static-First

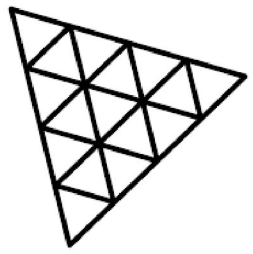
While there are dynamic elements, most Jamstack pages are pre-rendered meaning the front-end was compiled into HTML, CSS, and JS files like with our Gatsby/Sanity.io example.

Serverless Functions

Serverless functions are very beneficial because it takes away the need for the knowledge, time, effort, and cost of deploying and managing a server. If you need something done on the server-side, you can create a serverless lambda function and run that within Netlify or Vecel instead of having your own server. Obviously, this can’t be the solution for every application, but it can be for many.

High Performance, Secure & Scalable

Since everything is pre-rendered this makes **Jamstack sites extremely high performance**. Jamstack sites are also secure. There are not really any SQL injections or other database vulnerabilities, because again, everything is pre-rendered. Your costs are also far less because hosting static files is very cheap.



Animation, Graphics & Charts

Another part of front-end development is creating nice-looking graphics with animation as well as charts and other creative ways of displaying data.

This is another optional set of technologies. You are not required to learn any of this right now, but they can allow you to create some pretty cool stuff and are fun to work with. Let's look at some of the animation libraries that are available.

Motion UI

So we talked about CSS frameworks earlier and I mentioned Foundation. The company that created Foundation, *Zurb*, also created Motion UI which is a Sass library for creating transitions and animations.

It allows developers to rapidly prototype animated elements and seamlessly integrate them into websites.

You can install Motion with NPM, Yarn or Bower. The package includes a CSS file with a bunch of pre-made transitions and animations, along with the source Sass files, which let you build your own.

Framer Motion

Framer Motion is a production-ready animation library for React that is used to easily create animations. It has a pretty simple declarative syntax that is easy to read and maintain. Framer motion offers more advanced listeners and also extends the basic set of React event listeners.

There are quite a few options when it comes to React, such as **React Transition Group** and **React Spring**. So if you're a React developer, you may want to take a look at some of the different options.

Anime.js

Animie.js is a lightweight JavaScript library for animations and is really easy to use. It's pretty popular with 35K stars on Github. You can animate HTML, CSS, JavaScript, SVG, and DOM attributes. If you go to their website, you can see some really cool animations right on the homepage.

Three.js

Three.js is probably the most popular animation library. It has 50K to 60K stars on Github. It's extremely powerful and utilizes WebGL, which is a JavaScript API for rendering interactive 2D and 3D graphics. It allows GPU-accelerated usage of physics and image processing. So if you're looking to really get into powerful graphics and animation, I would suggest looking at Three.js.

Chart.js

We have quite a few ways to display data visually and Chart.js is an open-source JavaScript library for data visualization and supports all types of charts including bar, pie, line, area and radar charts. It's pretty easy to get started as well. There are all kinds of plugins, adapters and integration packages to extend it.

D3

D3 also known as "*Data-Driven Documents*" is a very powerful and popular data visualization JavaScript library. It makes use of HTML5, SVGs and CSS standards. It combines powerful visualizations and DOM manipulation to create really cool interactive elements.

GreenSock/GSAP

GSAP (GreenSock Animation Platform) is a robust toolset for creating very high-performance timeline-based animations. It uses "tweens" to give you more control over your animations. Even though it is a JavaScript library, you don't have to know a ton of JavaScript to start using it.

No-Code Tools

No-Code tools are GUIs (Graphical User Interfaces) to create websites and/or applications. They usually let you drag and drop UI elements like forms, menus, etc.

Before I talk about the specific tools, I want to talk about a couple of things. First, many developers do not like these tools because they look at them as a threat. Some feel that they will steal their jobs and render them useless. I completely disagree with this. I think in some cases, a business owner or someone that wants a website may choose to use this over hiring someone, but overall, I don't think the majority of the people looking for a website have the time to spend learning how to use one of these tools and frankly, they just do not want to. I think most would rather pay someone else to do it. Also, many of these are not easy to use. Sometimes I find website builders more difficult than actually writing code.

Second, I think that they can actually help developers and web designers. They are not only for non-tech people. They are also for us to use to build for clients. I know freelancers that use Webflow and barely know how to code that makes a six-figure income. So try not to look at these as threats but as just another tool.

Webflow

Webflow has an intuitive interface to build really nice-looking websites that are optimized for marketing. The websites are relatively fast and secure. It is also easy to collaborate and have different members of a team add content to the website.

Wix

Ah yes, the “W” word. Many developers hate Wix. Again, I just look at it for what it is and that is a tool. It is a tool for building websites without code. You can create brochure sites, blogs, etc. There are over 500 customizable website themes to choose from.

WordPress

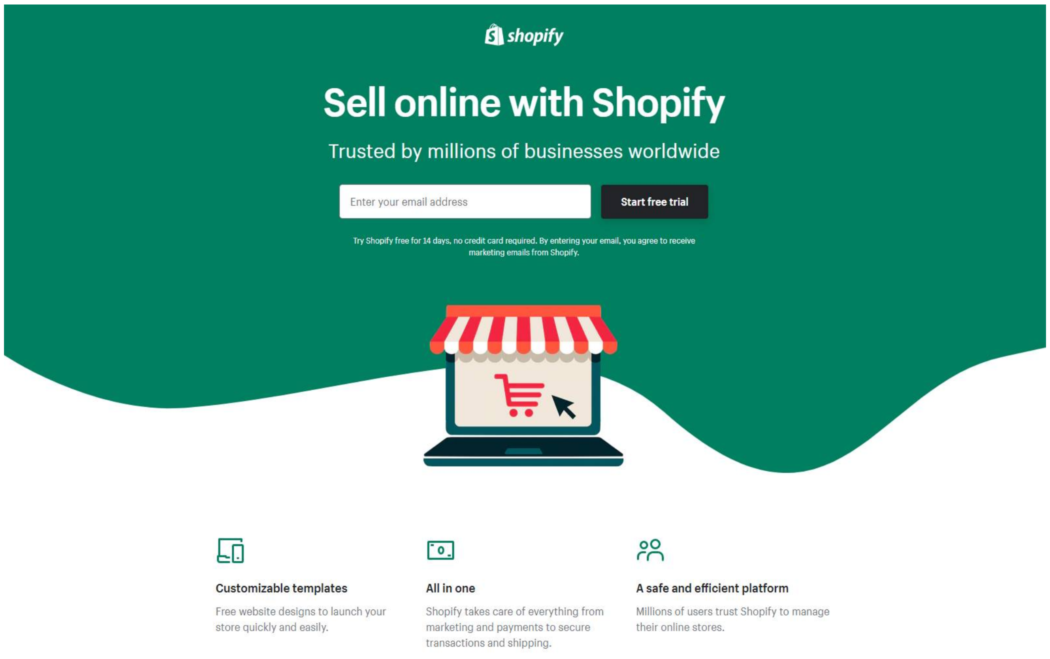
WordPress is a content management system and not really a no-code tool in the same way that Wix is. WordPress can be self-hosted and is fully open source. You can build and sell custom WordPress themes and plugins. Many developers build their own business using mostly or nothing but WordPress. It is a great tool for freelancers.

WordPress can also be used as a headless CMS. So if you want to have the admin-area functionality, but use something else for a front-end, you can do that. WordPress is not going anywhere for a long time.

Shopify

Shopify is an incredible platform in my opinion. It allows people (and yes, developers) to set up an online store fairly easily. I would compare Shopify more to WordPress than Wix or Webflow. Like WordPress, many developers make a good living building Shopify plugins and themes. Setting up an eCommerce site from scratch is no easy task. So for freelancers, Shopify is very useful.

There are so many paths to take in web development and web design. I think some people do not understand this. Working for a large company as a developer is much different than starting a smaller webshop and serving small business customers. Each will work with many different technologies and will have many different approaches. When you see people online making fun of WordPress or Webflow, chances are they are beginners that just like to sound smart or they have only had the large company experience and do not understand what it's like to have to get projects out quickly to clients and being responsible for every aspect of the website or application.



Sell online with Shopify

Trusted by millions of businesses worldwide

Try Shopify free for 14 days, no credit card required. By entering your email, you agree to receive marketing emails from Shopify.

Customizable templates

Free website designs to launch your store quickly and easily.

All in one

Shopify takes care of everything from marketing and payments to secure transactions and shipping.

A safe and efficient platform

Millions of users trust Shopify to manage their online stores.

Visit https://shopify.com for more info

Front-end Superstar

So we've talked about a lot. I would say that at a high level, we talked about pretty much everything that a front-end developer would do. I'm sure to many of you, this is really overwhelming. I want to encourage you to just think about all of these as the options that are out there. You're not expected to know or to learn all of this. Also, once you learn a front-end framework like React, learning something like Next.js or Gatsby isn't all that difficult. They're all in the same ecosystem. It's not like learning completely different programming languages or frameworks.

As a front-end superstar, you should be able to do the following.

✔ Front-end Framework

By now, you should be able to build user interfaces with a front-end framework. Which one you use is up to you. There are four really popular and well-established frameworks. You should also know some of the technologies within that framework's ecosystem.

✓ Work With State

You should understand the concept of "state" pretty well. This includes your component-level state like form fields or if a menu is open or not as well as global or app level state like data coming from a database/API. You may use something that is included with the framework such as React's context API or a 3rd party state manager like Redux or Vuex.

✓ REST API & HTTP

You should understand everything about HTTP when it comes to making requests to a server and then getting a response back. You should know how to work with REST APIs, HTTP status codes (at least the main ones), HTTP headers and so on. We will have more on this in the next section.

You may even work with GraphQL. I saved that for later because in addition to consuming GraphQL APIs, you create them on the server.

Added Skills

These are skills that are not necessarily required, but they are useful and can help you on your resume.

✓ TypeScript

You may know how to use static-typing with TypeScript. Also things like decorators and interfaces. This is optional but is getting more popular every year and some points will be required.

✓ SSR (Server-Side Rendering)

I think working with server-side rendered pages is going to become more and more popular as well with frameworks like Next and Remix. It is definitely worth it to learn and it's not very difficult if you already know the core framework.

✓ Jamstack & SSG

The Jamstack is also growing in popularity using static-site generators like Gatsby and using a headless content management system for data to build very high-performance websites.

✓ Testing

You may do some testing with a framework like Jest. People seem to learn testing at their own pace when they see fit.

✓ No-Code Tools

You may also look into using some no-code tools to help you build websites quicker. As a developer, you would most likely learn WordPress or Shopify over something like Wix.

CHAPTER 4

BACK-END DEVELOPMENT

In this chapter we will look at technologies and concepts that you need to know as a back-end or full-stack developer. This includes programming languages, back-end frameworks, database systems and more.

Server-Side Programming Languages

Up to this point, we have been talking about technologies that are used mostly in the front-end/UI. Now we're going to start looking at some server-side or back-end technologies.

First, we have programming languages. JavaScript is the language of the browser. That's why we have talked about it so much in the front-end sections. The server can basically run any language that you want.

There are different languages that are good for different things. For instance, Python is great for machine learning and AI. C++ is great for game engines and things of that sort. When it comes to web development, which is dealing with HTTP, working with databases, etc, these are relatively easy tasks for the computer. For that reason, **just**

about any programming language can be used for certain aspects of web development. This gives you a lot of choices.

Let's take a look at some of the popular programming languages that can be used in web development. I will also display a very simple "Hello World" to show stuff like variable syntax, print functions, etc.

Node.js

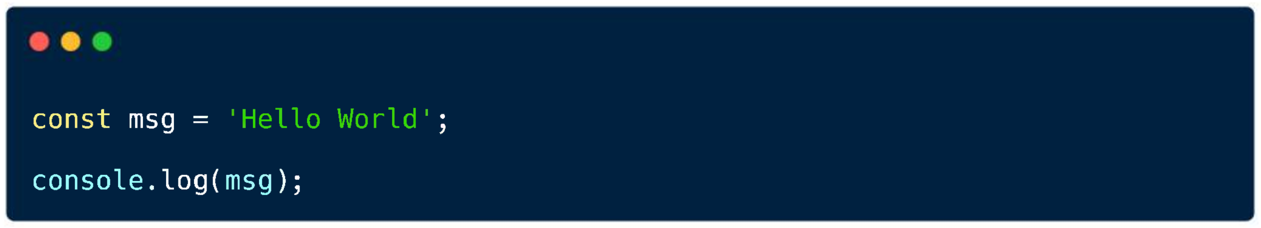
Node.js is not a "language", it is a **JavaScript runtime** that was released in 2009. It essentially allows JavaScript to run on the server as opposed to running in the browser (front-end). It has become extremely popular in full-stack web development as it is great for creating APIs and microservices. It also includes NPM (Node Package Manager) with a rich ecosystem.

Node is very high-performance for many tasks. It is built with a **non-blocking, event-driven architecture**. By non-blocking, I mean that it doesn't block further operations from happening when working on a task. Non-blocking methods are executed asynchronously, meaning that the program may not execute line-by-line. The program creates a callback to be run on completion. This is a huge advantage for many (not all) types of tasks.

Because of the way Node.js works, it is great for things like APIs, microservices, real-time applications, etc. Where it falls short is CPU-heavy tasks with a lot of calculation.

Node.js is my first choice because it is great for the types of applications that I build and I like having the same language (JavaScript) on both the front and back end.

Hello World Example: **Node.js / JavaScript**



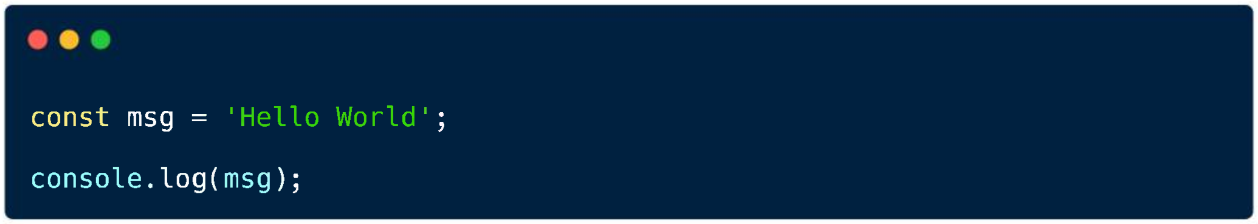
```
const msg = 'Hello World';
console.log(msg);
```

Deno

There is another option if you want to use JavaScript on the server and that is Deno. It was created by Ryan Dahl, who is also the creator of Node.js. When Deno was announced, there was a ton of hype. Articles and videos were created saying it would make Node.js extinct. However, that hype completely died down and now nobody really talks about it.

I do think that Deno has a place in web development in the future because it does address some of the issues with Node and has some really great features including TypeScript by default, built-in ES Modules, promises, better security and more. It's just very immature right now so I would not suggest choosing it as your first server-side technology to learn, but don't be surprised if we see Deno rise to the top in the future.

Hello World Example: **Deno / JavaScript**



```
const msg = 'Hello World';
console.log(msg);
```

Python

Python is an amazing language. It is used in so many areas of technology. It is exceptional for things like **data science** and **machine learning**, but it is also great for web development. One of my favorite frameworks regardless of language is Python Django. We will talk about back-end frameworks soon.

Python is very popular. It has an easy-to-use syntax that is almost English-like. It is quite different from C-syntax languages in that it doesn't use curly braces or semicolons, it uses spaces. IT can feel weird at first coming from a language like JavaScript, but you get used to it really fast. Like JavaScript, it is an interpreted and dynamically-typed language. Python uses the **PIP package manager,** which is similar to NPM with JavaScript. So you can install all kinds of 3rd party packages into your environment.

I would suggest Python if you think you will ever do some of the other things that it is known for including data science, machine learning & AI, automation, etc. This includes building web apps that utilize these things. You may also want to check out some of the web development frameworks that Python offers like Django and Flask.

Hello World Example: **Python**

```
msg = "Hello World"

print(msg)
```

PHP

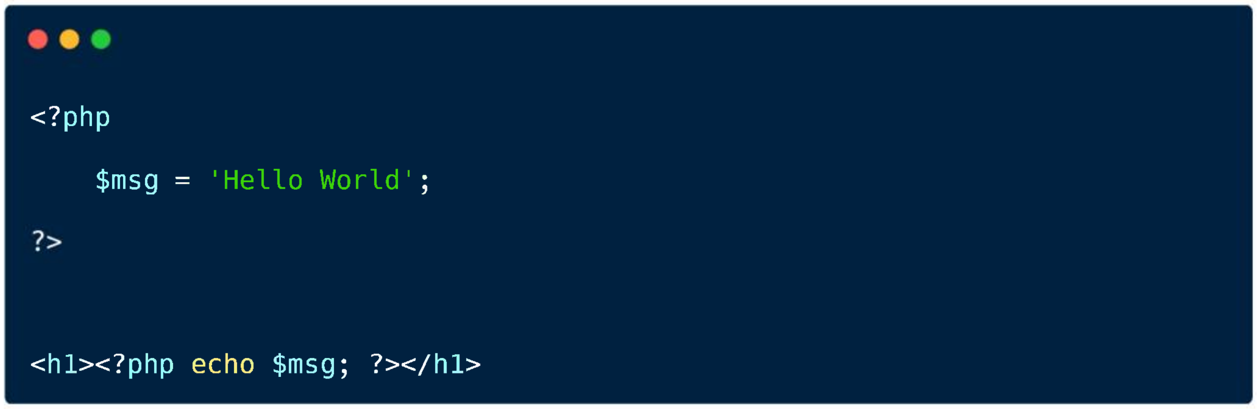
PHP is great for what it does and that is practical web development. PHP seems to get a lot of hate, but most of it is unwarranted and from people that have barely used it. It has become sort of a meme to bash PHP. Don’t let that discourage you from using it though. I know many people that make a lot of money working with PHP, especially freelancers and small firms.

PHP is a very practical language for web development. It is easy to deploy, as you can simply upload .php files to your web server and they will get parsed and displayed in the browser. In addition to that, you can put HTML directly into a .php file and it will get rendered as if it were a .html file.

PHP also has a ton of great software for developers. WordPress is a popular CMS that we talked about earlier. There are also eCommerce platforms like Magento, which may not be the most high-performance, but they offer a lot of functionality out of the box. This lets people build stuff very quickly, which is why it is great for freelancers. If your goal is to get a job as a developer, you may want to look at some other options.

PHP also has the *Composer package manager* to easily install packages for just about anything including *Laravel*, which is another one of my absolute favorite frameworks.

Hello World Example: **PHP with HTML**



```
<?php
    $msg = 'Hello World';
?>

<h1><?php echo $msg; ?></h1>
```

C#

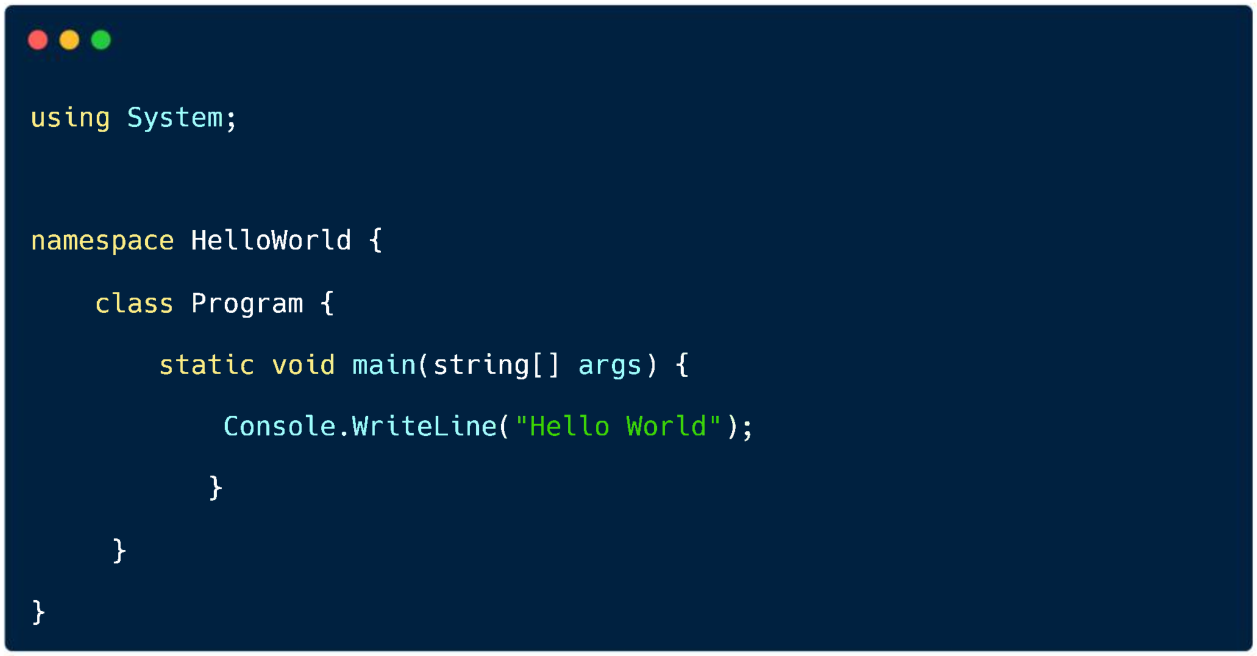
C# is a great general-purpose language. It can be used for many things. In fact, my own experience with C# is not with web dev, but with Windows desktop applications.

As far as web development goes, it is used a lot in the big business world. If you use C#, you will probably use the Visual Studio IDE as opposed to Visual Studio Code. The IDE offers a lot for C# devs.

Many C# users use the .NET framework, which is very powerful and extremely popular. There is a web framework called **ASP.NET MVC** that many web developers use.

C# is an OOP (Object Oriented Programming) language and is strongly typed. It can be thought of as both a compiled and interpreted language. It is compiled to a virtual language and then interpreted by a VM. Visual Studio has all of the tools that you need for writing C# apps integrated within it.

Hello World Example: **C#**



```
using System;


namespace HelloWorld {
    class Program {
        static void main(string[] args) {
            Console.WriteLine("Hello World");
           }
     }
}
```

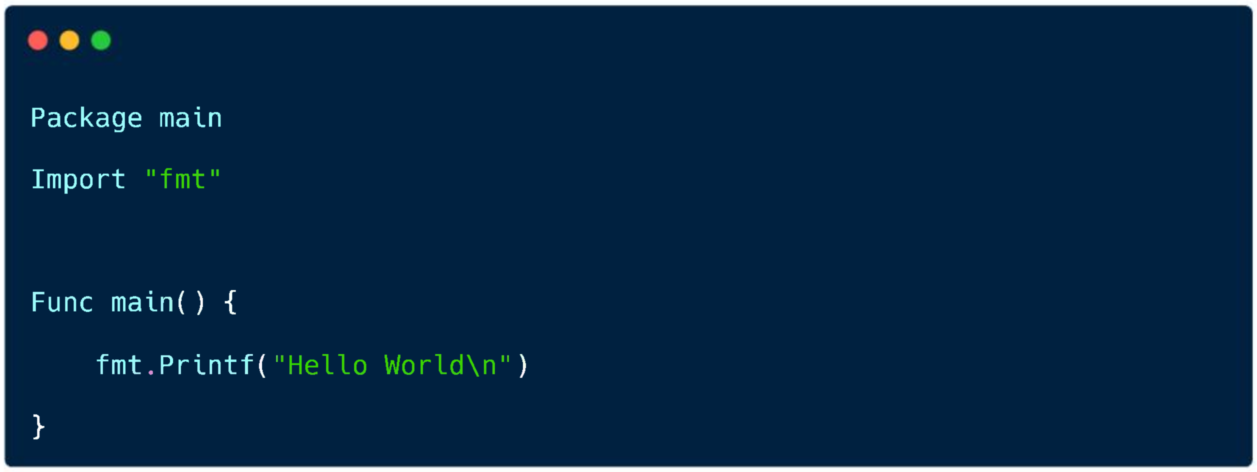
GoLang

Go or “GoLang” is an extremely powerful language created by Google. Go is another language that is popular in enterprise web development. It is highly scalable and high-performance out of the box. It is a compiled, strongly-typed language that can run locally or in the cloud.

You actually don’t need a 3rd party server like Apache or NGINX to serve content over the web. You can build web servers directly with Go that are more performant than something like NGINX.

You also don’t really need a specific framework, as you would use with most other languages. There are a lot of native tools and a great standard library to work with HTTP, HTML, JSON, etc. With that said, there are some 3rd party frameworks for GO that I will talk about later.

Hello World Example: **GoLang**



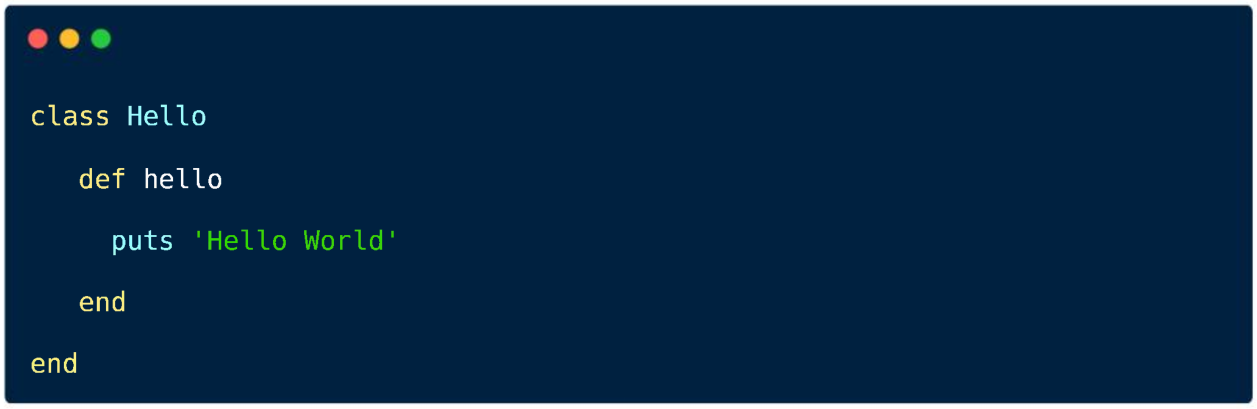
```
Package main
Import "fmt"


Func main() {
    fmt.Printf("Hello World\n")
}
```

Ruby

Ruby is an interpreted language that I worked with for a couple of years and I really like it. The syntax is very clean and almost like writing in English. When we talk about Ruby for web development, we’re usually talking about Ruby on Rails, which is a web framework. We’ll talk about frameworks next.

As far as popularity, Ruby has fallen a few notches in the last few years, however, there are still companies using it and if you find you really like the language, I wouldn’t discourage you from learning and using it.

Hello World Example: **Ruby**

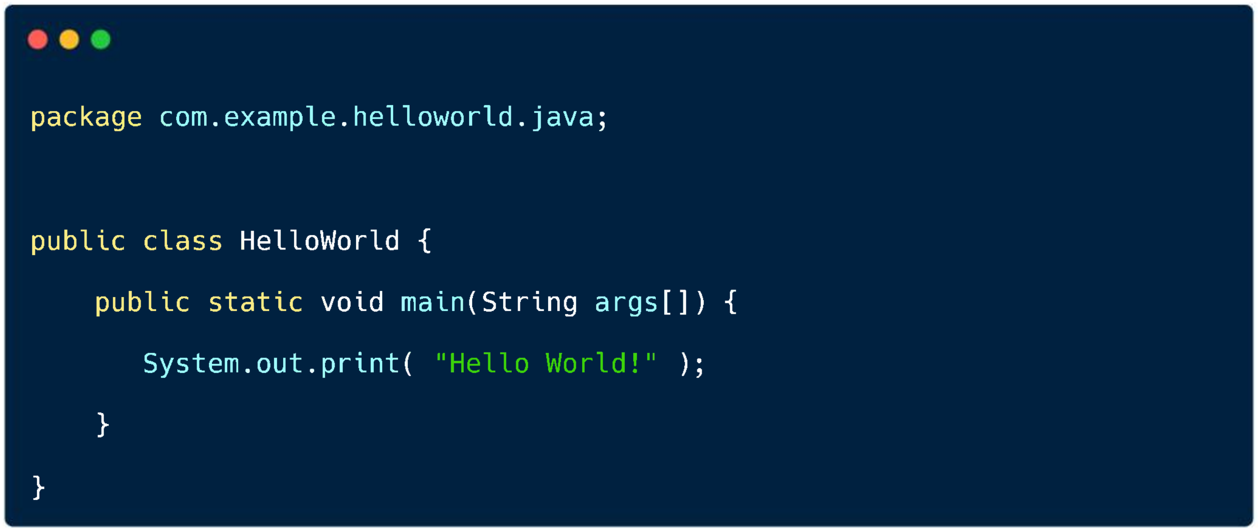


```ruby
class Hello
   def hello
     puts 'Hello World'
   end
end
```

Java

Java is another language that is used in a lot of ways. In addition to web dev stuff, it is used for things like desktop and mobile applications. It’s really big in the enterprise world. The Spring Boot framework is pretty popular. I personally think Java is a very clean language. It was actually my first introduction to programming ever and I think it’s a pretty good language to learn the fundamentals of programming. I wouldn't personally use it for web development, but that’s just my own preference.

Hello World Example: **Java**



```java
package com.example.helloworld.java;


public class HelloWorld {
    public static void main(String args[]) {
       System.out.print( "Hello World!" );
    }
}
```

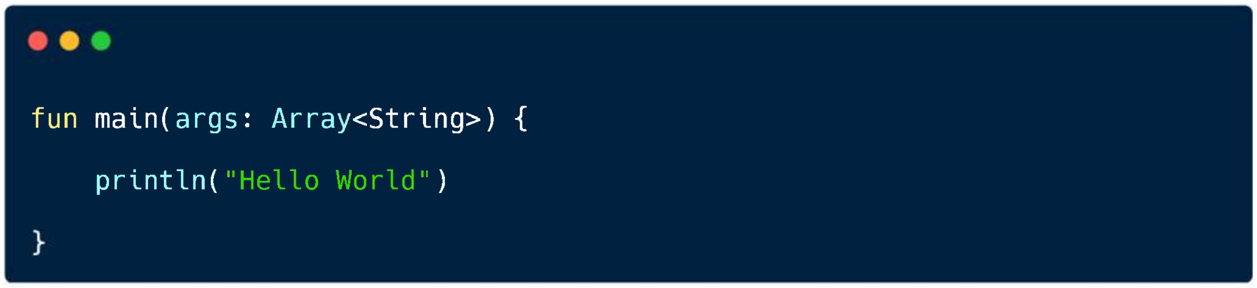
Kotlin

Kotlin is a relatively new language and was actually designed for the **JVM (Java Virtual Machine)**, so in many cases, Kotlin is replacing Java.

Kotlin is actually 100% interoperable with Java, so you can have as little or as much Kotlin code as you want in your project.

Kotline is widely used in the Android app world. It can also be used for web apps. It has a pretty smooth learning curve and it’s very scalable.

Hello World Example: **Kotlin**



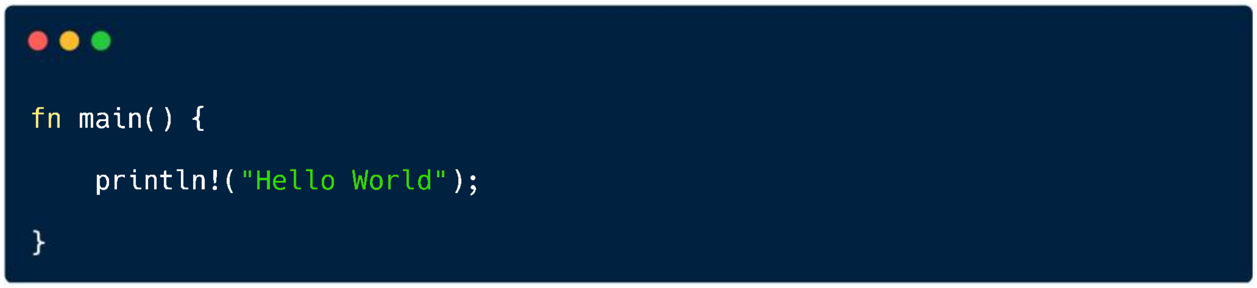
```
fun main(args: Array<String>) {
    println("Hello World")
}
```

Rust

Rust is a fairly new lower-level language that is very powerful and used for many different things including backend web development. When you compare it to other low-level languages like C and C++, it is relatively easier to learn, especially when it comes to memory management.

One really cool thing about Rust is that it can be used with something called Web Assembly. I do have a section later on where we talk about web assembly, so I just want to focus on back-end web development right now and Rust is a viable option. It uses Hyper as a core HTTP server and can be great for APIs and microservices.

Hello World Example: **Rust**

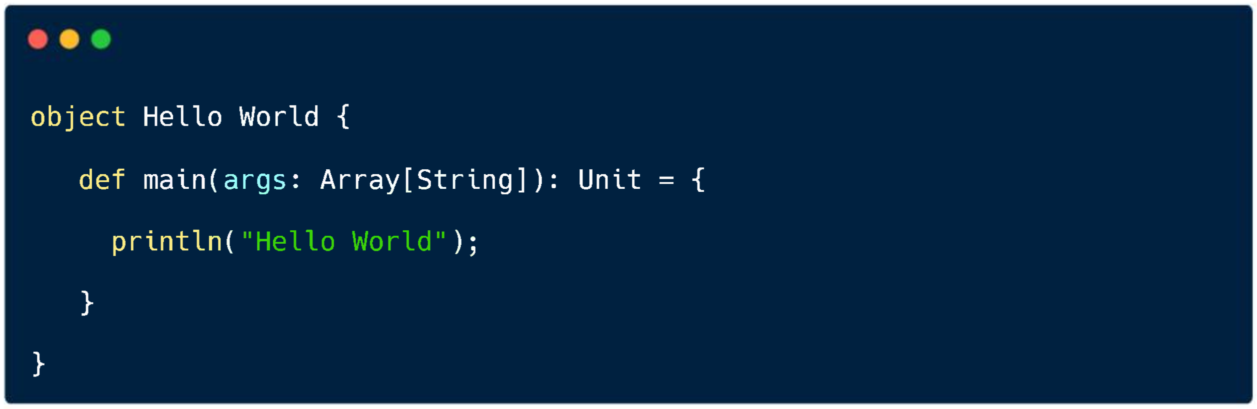


```
fn main() {
    println!("Hello World");
}
```

Scala

Scala is another option for back-end web development. Like many other languages, it combines object-oriented and functional programming. It powers the data engineering infrastructure of many companies, but can also be used for building APIs and microservices. The Play framework is pretty popular and can be used with both Java and Scala.

Hello World Example: **Scala**



```
object Hello World {
   def main(args: Array[String]): Unit = {
     println("Hello World");
   }
}
```

R

R or Rlang is not a language that I have any experience with at all. I have done some research on it. I also know other developers that use it. It is a fantastic language for statistical computing, data analytics and scientific research. In some ways, it does a lot of the stuff that Python does really well. So if you want to get into data analytics, R may be a great choice. For web development, you could use R with Markdown and the Knitr engine. There's also a package called "Shiny" that lets you create web pages. That's about all I know as far as R programming.

Hello World Example: **R**

```
hello <- "Hello World"
print (hello)
```

Swift

Swift is a popular general-purpose programming language, but it is mostly known for developing mobile apps. In particular, native iOS apps along with Objective-C. Swift can be used for web development, so I did want to add it here, however, there aren't many tools available when it comes to web dev, so I would only suggest learning Swift for mobile development at least at this point.

Hello World Example: **Swift**

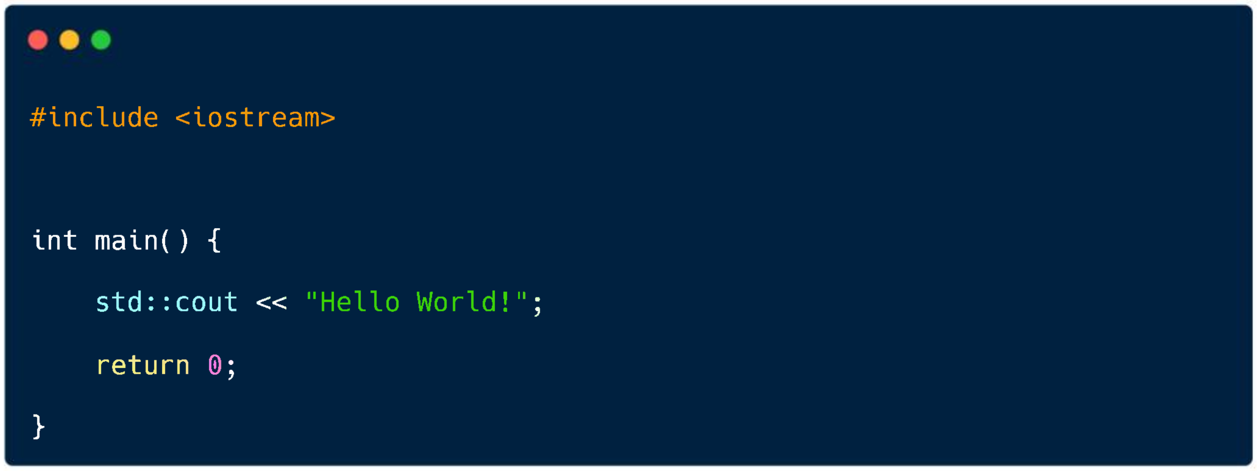
```
Import Swift
print("Hello World")
```

C & C++

Both C & C++ are much lower-level languages that are compiled. They are not interpreted like most of the languages above. This means the code has to run through a compiler. They are not typically used for what we would call “web development” as in building websites. However, they are extremely powerful and you could do stuff like create modules for Apache or Nginx.

There are in fact web frameworks such as CppCMS, however, I have not heard much about them at all, so I don’t have much to say about them.

Hello World Example: **C++**



```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Server-Side Frameworks

As with the front-end, we also have web frameworks for the back-end and every language has them. Web frameworks are designed to handle HTTP requests and send a response. The response could contain HTML rendered on the server or it can return JSON, which is commonly used when creating APIs to be consumed by the front-end.

Opinionated vs. Non-Opinionated

Some frameworks are more opinionated than others. What I mean by that is some of them require you to build in a certain way when it comes to your file structure, naming conventions, etc. Opinionated frameworks are often much more high-level.

The advantage of opinionated frameworks vs. more minimalistic is that you get much more out of the box. For instance, Python Django is pretty opinionated and high-level and you get an entire admin area with CRUD (create, read, update, delete) functionality on all of your models. This is because you write your code and structure it in a certain way. The downside is that you don't have as much freedom and customization as you would with a lower level and less opinionated framework like Express or Python Flask. With minimalistic frameworks, you will use more 3rd party packages for certain aspects of your application. The type of framework that you choose usually comes down to your preference and also depends on what you're creating.

MVC (Model View Controller)

MVC is a very popular design pattern that many of the frameworks that I will be mentioning use. I would say the majority of them use MVC in one way or another.

The idea is to separate your code into 3 logical components (model, view and controller). Each of these components handles a certain part of your application. I won't go into huge detail with MVC, but give you a brief summary of what each component does.

Model

The model is typically **the part of your code that deals with the data**. Your data can come from anywhere, but it typically comes from a database. If you need to fetch a list of employees from the database, it is typically the job of the model to do so. You may also use an ORM (Object Relational Mapper) or ODM (Object Data Mapper) to help create your models. For instance,Laravel uses an ORM called *Eloquent* to create and work with models.

View

The view is just that. It is **the view or the user interface display** of the application. These could be views that are rendered on the server. Laravel uses a template engine called Blade for views. Django commonly uses Jinja and Express could use ESM or Jade. Another popular option is to use a UI library like React as the view. In fact, React is often called the “V” in MVC.

Controller

The controller usually acts as an **interface between the model and the view.** The controller asks the model for data and then loads the view and passes the data to it. Typically in web apps, you create “routes”, which are URLs that the browser or an HTTP client hits and specific routes will call specific methods in the controller class.

Some more opinionated frameworks have a very strict MVC folder structure and more minimalistic ones give you the freedom to create your own implementation.

Let’s take a look at some of the most popular web frameworks for each language.

Node.js Frameworks

Node has a ton of web frameworks. There are way too many to list all of them here, but these are some of the ones that I think are worth mentioning.

Express

Express is definitely the most popular Node.js framework. There are also a lot of frameworks built on top of Express. It is a very *minimalistic* framework, which means that there is not a bunch of stuff included and it is not very opinionated. This gives you complete freedom to build and structure your code and files how you want. You can utilize whatever extra packages that you want to add specific functionality. For instance, Bcrypt is a popular package for hashing passwords. Mongoose is a popular ODM (Object Data Mapper) for working with a MongoDB database. So when you use Express, expect to use many other packages with it.

Companies like Twitter, Uber and IBM deploy apps built on Express.

Fastify

Fastify is a newer framework for building web apps and APIs. It's something that I started working within 2021 and really liked and I could see myself using it in production. It's super-fast, as the name implies. It also has a powerful plugin architecture, with documentation tools and much more.

I would say that Fastify is pretty minimalistic, but has more included in it than something like Express. In addition to the plugin system, it is TypeScript friendly, has extensive logging and debugging tools and more.

Koa

Koa is another very minimalistic option. In fact, it was created by the creators of Express and is very similar. The syntax is even similar. It aims to be a simpler, smaller and more expressive and robust foundation for web apps and APIs.

Nest.js

Nest.js is a scaleable, progressive framework and offers a lot out of the box. It runs on top of Express but can also be configured to use Fastify and others. It uses a level of abstraction, but can also expose Express, Fastify, etc directly to the developer.

Nest is built for the enterprise. It includes TypeScript by default and really reminds me of Angular but on the back-end. There’s a lot of the same structure. So I think if you’re an Angular developer, Nest.js is definitely something that you should check out.

Adonis

Adonis is a high-level, feature-packed framework that really reminds me of Laravel for PHP. It has migrations and a bunch of other tools, so if you like an all batteries included type of framework and you're a JavaScript developer, Adonis is a great fit for you.

Python Frameworks

Like I said earlier, Python is great for a lot of things other than web development, but there are some great frameworks for Python web developers.

Django

Django is definitely one of the best web frameworks out there in my own opinion. You get so many features out of the box such as an entire admin area. Django allows you to build pretty complex apps and APIs in a short amount of time. It is very specific, in that you have to do things the "Django way", but you get a lot for it. It's very fast and scalable as well. I barely knew Python when I learned Django, but it is so easy to work with, you don't really need to know much of the Python language, you simply follow the conventions.

Django is what is known as an "MVT (Model View Template)" framework, which is similar to MVC (Model View Controller). The main difference is that Django itself takes care of the controller aspect, leaving us with the template.

You can build REST APIs pretty easily with packages like *Django REST framework.* You can also of course use templates and render pages on the server.

Flask

Flask is a framework that I like almost as much as Django but it's completely different. Flask is very minimal and feels a lot like Express. You can do things your own way but it gives you some helpful tools to create routes and work with the HTTP request and response cycle.

Picking between Django and Flask really depends on the project and the type of developer that you are. If you need a lot of freedom to structure the project how you want and use a bunch of packages, etc, then Flask is a great choice.

FastAPI

FastAPI is a newer framework for building well… fast APIs. It's very high performance and on par with Node.js and GoLang. It's fairly easy to use as well.

PHP Frameworks

PHP is a bit different than some of these other languages because it's basically a template language itself. You can run a PHP page directly in the browser. It does however have some nice frameworks.

Laravel

Laravel is extremely popular. Sometimes PHP gets a bad wrap, but it seems Laravel gets a lot of respect in the developer community. PHP can be a messy language, but Laravel is actually really elegant and offers just about as much as any framework can including the Artisan CLI tool, migrations, authentication, templates, validation and so much more. If you're a PHP developer, I would highly suggest Laravel.

Symfony

Symfony is a set of reusable PHP components and it’s important to mention that Laravel is actually built on Symfony components. You can use Symfony by itself, but it is in my opinion, one of the more difficult frameworks to learn. It is extremely powerful though.

SlimPHP

I wanted to put SlimPHP on this list, not because of its popularity, but because I really like it. Unlike Laravel it gives you a minimalist option like Express but for PHP. It’s very lightweight and easy to use. It is great for creating relatively simple REST APIs.

C# Frameworks

ASP.NET

.NET is commonly used with C#. .NET is a very popular developer platform by Microsoft, that is made up of tools and libraries for building all kinds of applications. ASP.NET extends the .NET platform with tools and libraries for building web applications. So for C#, you’re probably going to be looking at ASP.NET.

ASP.NET MVC is a model-view-controller framework that sits on top of ASP.NET and uses APIs from it.

Blazor

Blazor is an open-source framework, also created by Microsoft, that enables developers to create web apps using C#. What is interesting about Blazor, is that it is used for both back-end as well as front-end. It allows the front-end logic to run directly in the browser via C#, HTML and CSS.

I have yet to use Blazor, but it is definitely something I would like to look further into.

GoLang Frameworks

GoLang has a large standard library, so in many cases, you may not even need to use a framework. However, there are frameworks available for creating web apps.

Gin

Gin is the most popular Go framework and is very straightforward and easy to use. It is also very performant. The router in the Go standard library can be pretty limiting, Gin uses Radix tree-based routing.

Beego

Beego is a framework used for rapid development of enterprise applications in Go, including REST APIs, web apps and back-end services. It was inspired by Flask, Tornado, which is another Python framework and Ruby Sinatra.

Ruby Frameworks

Ruby on Rails

If you're using Ruby for web development, you're probably using Ruby on Rails. My personal opinion is that ROR is one of the best MVC frameworks ever created. The rapid development is crazy. ROR has powerful command-line tools out of the box, migrations, scaffolding, templates, localization and much more. It's not as popular as it once was in the industry, but it is still one of the best and most put-together frameworks in my opinion. It has influenced many other frameworks such as Laravel and Django.

One big thing that Ruby on Rails tackles really well is the Don't Repeat Yourself (DRY) principle. It helps eliminate tedious and repetitive tasks like creating forms, menus and tables. It also uses Convention over Configuration (CoC), which is a software concept that aims to reduce the number of decisions the developer has to make by following certain conventions and not having to do too much configuration.

So with that said, if you are looking to find a job in the industry, there may not be too many companies looking for Rails developers, but if you are building your own projects or freelancing, it may be a good framework to look at.

Sinatra

Sinatra is the second most popular framework for Ruby. It's a much smaller framework and is sometimes referred to as a "microframework". I think that it is a good minimalistic alternative to Rails for Ruby developers.

Java Frameworks

Spring

Spring is very popular in the Java world. It is actually a programming model and ecosystem that is built on Java and has several parts such as *Spring Boot*, which makes it easy to create stand-alone applications. *Spring Cloud* is a combination of components that allows developers to build cloud-native apps. There is a lot to Spring and it is an extremely powerful framework.

Struts

Struts is another Java framework for building web apps. It follows the MVC design pattern and extends the JSP (Java Server Pages) API. Struts provide more flexibility over the traditional MVC approach using servlets and JSP alone.

Kotlin Frameworks

Spring

Spring 5.0 started supporting Kotlin. So you can use the Spring framework with both Java and Kotlin.

Vert.x

Vert.x is a framework for Kotlin that is very flexible and resource-efficient. It’s asynchronous with callbacks and promises and it has its own ecosystem as well.

Scala Frameworks

Play

Scala actually uses the Play framework which can also be used with Java. Play includes powerful console and build tools, testing tools and it scales predictably. Play is used in startups as well as the enterprise level.

Lift

I have never used Lift, which is a Scala framework that claims to be the most secure web framework available. Lift apps are high performance, scalable and easy to maintain.

Rust Frameworks

Rocket

Rocket allows you to write fast, secure web apps with Rust. It’s a type-safe framework that is easy to use. I looked at some of the syntax and even though I’m a beginner to Rust, it was pretty clear how to create endpoints, etc.

Actix

Actix is another framework that looks very interesting. Again, I think the way you build with this framework is similar to Express. It seems like the code is pretty straightforward and minimalistic. Rust is definitely something that I want to get deeper into.

Databases

Back-end and full-stack developers need to work with a database to store data. There are different types of databases, but I would say the most popular are **relational databases**. There are also **NoSQL databases** that are pretty popular as well. Relational databases store data in tables and rows and typically use Structured Query Language (SQL) to create, read, update and delete data. So SQL is probably something that you'll be learning as a back-end or full-stack developer. NoSQL databases typically store data in "collections", which are usually formatted like JSON objects.

Like with everything else, it's going to be preference. Here are some of the most common database systems used in web development.

PostgreSQL

Postgres is my favorite relational database. It’s actually an **object-relational database** as opposed to MySQL, which is a purely relational database. Postgres has features like table inheritance and function overloading. Postgres also adheres more closely to SQL standards. Even though all the popular relational databases use SQL, the way that they implement it can vary.

MySQL

MySQL is another popular relational database. In many web apps, you won’t see much of a difference if you use Postgres or MySQL. Especially small to medium-sized apps. So it’s completely up to you. MySQL seems to be really popular in the PHP world.

MS SQL Server

So SQL Server is Microsoft's implementation of a relational database with SQL. I personally have never used it, so there isn’t too much I can elaborate on.

SQLite

So SQLite is a much lighter file-based database that isn't really used in production, however, it is a great development database. You don't have to go through a crazy install process, you just have a *something.db* file and you can then create, read, update and delete data using that file. I think it's a very valuable tool for development.

MongoDB

So MongoDB is a completely different type of database. It is a NoSQL database, but with NoSQL, there are sub-categories and MongoDB is a *document database*. It holds collections of documents, rather than tables of records. Each document is basically a JSON object. In fact, Mongo uses BSON, which stands for Binary JavaScript Object Notation and extends JSON with some extra types such as dates.

MongoDB is really popular in the Node.js world. You've probably heard of the **MERN stack**, which stands for MongoDB, Express, React and Node.js. You also have **MEAN** and **MEVN** for Angular and Vue. Since MongoDB formats the data like a JavaScript Object, it's easy for JavaScript developers to understand. MongoDB also has a cloud version called

MongoDB Atlas, which is very popular and I would suggest using it over a local installation in most cases.

Firebase

Firebase is different from the databases that we have talked about already. Firebase is an entire hosted platform by Google. It does offer a NoSQL database, but it also offers authentication, file storage, serverless functions and more. It's an all-in-one platform and I really like it for small to medium-sized projects. The downside is that you don't have full control of the database. It's not something that you install on your own server. If Google decides to just drop Firebase, you could be in trouble. I think the chances of that happening are really small, but it is something to think about. I personally love it.

Supabase

Supabase is a new alternative to Firebase. It includes a database, but it actually uses a Postgres Database as opposed to Firebase, which uses a NoSQL document database. It has an abstraction layer so that you don't have to write tedious SQL queries. It also includes authentication and some other things, so it's another all-in-one platform. It's completely open-source and if it did go under, you could simply backup your Postgres database and move to a new server. That is definitely a plus, compared to Firebase, which is locked in with Google services.

Redis

Redis is also very different from the rest. Redis is an in-memory data structure store that is often used for caching**.** It can be used with multiple stacks. It has different data structures like strings, lists, sets and so on. It has great performance and can be used for caching, session management, real-time analytics or as a general database. It's a great tool.

More on SQL (Structured Query Language)

SQL is important for most developers that work on the back-end. Here are some examples of what SQL looks like

Selecting columns from a table

```
SELECT column1, column2 FROM table_name;
```

Inserting data into a table

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Updating data

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Deleting data

```
DELETE FROM table_name WHERE condition;
```

ORMs (Object Relational Mappers)

No matter which database you choose to learn, you'll also have an ecosystem of tools to learn along with it. There are tools like MySQL Workbench, PGAdmin for Postgres, MongoDB Compass, these are all used to access and manage your database directly. In addition to these tools, you also have something called an ORM or Object Relational Mapper. The way that I like to describe an ORM or an ODM (Object Data Mapper) for document databases is basically **a layer that sits on top of your database that you can use to query and manipulate data.** You do have SQL for relational databases, but SQL statements are not very fun to write. ORMs to make it easier to interact with the database.

Some frameworks have built-in ORMs such as Laravel and Django. There are also some standalone ORMs. Many of which are for specific languages and can use multiple databases. When using Node.js with a relational database, I like *Sequelize*. When using MongoDB, I almost always use *Mongoose*. *Prisma* is a great ORM that I just started working with. A combination of Prisma and an SSR like Next.js or Remix along with a Postgres database makes for a really nice tech stack. You also have *SQLAlchemy* for Python and *Doctrine* for PHP. There are more for every language. So this is something that you'll learn more about as a back-end or full-stack developer.

Typically, you will create models like "Post" and then you will have methods available such as "find()". Then to fetch records, you can do something like

```
posts.find(1)
```

As opposed to...

```
SELECT * FROM posts WHERE ID = 1
```

REST API

REST APIS & HTTP

Understanding and building REST APIs is something that you should learn early on as a **back-end** or **full-stack developer**. REST APIs conform to the REST architectural style and allow interaction with RESTful services.

REST stands for **Representational State Transfer**. We usually make HTTP requests to work with data or resources on the server that are typically stored in a database. Things like users, posts, etc.

We access resources from an HTTP client, which could be a standalone program such as **Postman** or your application's front-end. We would make a request to the APIs endpoints, which are URLs that are structured in a very specific way using specific HTTP methods. Let's look at some example endpoints.

```
GET  somedomain.com/api/todos
```

A GET request usually means that you're fetching some kind of data. You’re not manipulating it, you’re just getting it. In fact, when you go to a regular web page, you make a GET request and the server gives you an HTML document. With most REST APIs, you get back JSON data. You can then use that data in your front-end application, which might be React or Vue or whatever else. So making a GET request to the endpoint above, you should get all todos.

```
POST    somedomain.com/api/todos
```

A POST request means you are posting/adding data. This would add a new post. You would send the todo data in the body of the HTTP request.

```
PUT  somedomain.com/api/todos/100
```

A PUT or PATCH request is used to update data, so this endpoint would update the todo with the ID of 100. You would send data in the body.

```
DELETE   somedomain.com/api/todos/100
```

A DELETE request to delete the todo with the ID of 100.

As a back-end developer, you would create these endpoints and fetch the todos from the database and send them in the response. As a front-end developer, you would make the request using an HTTP client and use the response to add the data to the user interface. As a full-stack developer, you'll do both.

HTTP Status Codes

You will also need to know the common HTTP status codes. These indicate whether or not the request has been successfully completed. There are 5 different ranges of codes that represent certain responses.

100 - 199	= Informational
200 - 299	= Success
300 - 399	= Redirection
400 - 499	= Client Error
500 - 599	= Server Error

These are the ranges. Let's look at some actual codes.

200 - Successful. Everything went as planned. You would usually get some kind of payload in the response body
201 - Successful and something was created
204 - Successful but no content to respond with
301 - Moved permanently. The URL of the request has changed
304 - Nothing has changed (caching purposes)
400 - Bad request. Something is wrong with the request from the client. This could be missing information, etc
401 - Unauthorized. You are not logged in and should be
403 - Forbidden. Client does not have the access rights for this route
404 - Not found. The server can not find the resource you are looking for
405 - Method not allowed. The endpoint does not allow the method (POST, PUT, etc) that you are using
500 - Internal server error. The server encountered an issue.

These codes are fundamentals for a back-end developer.

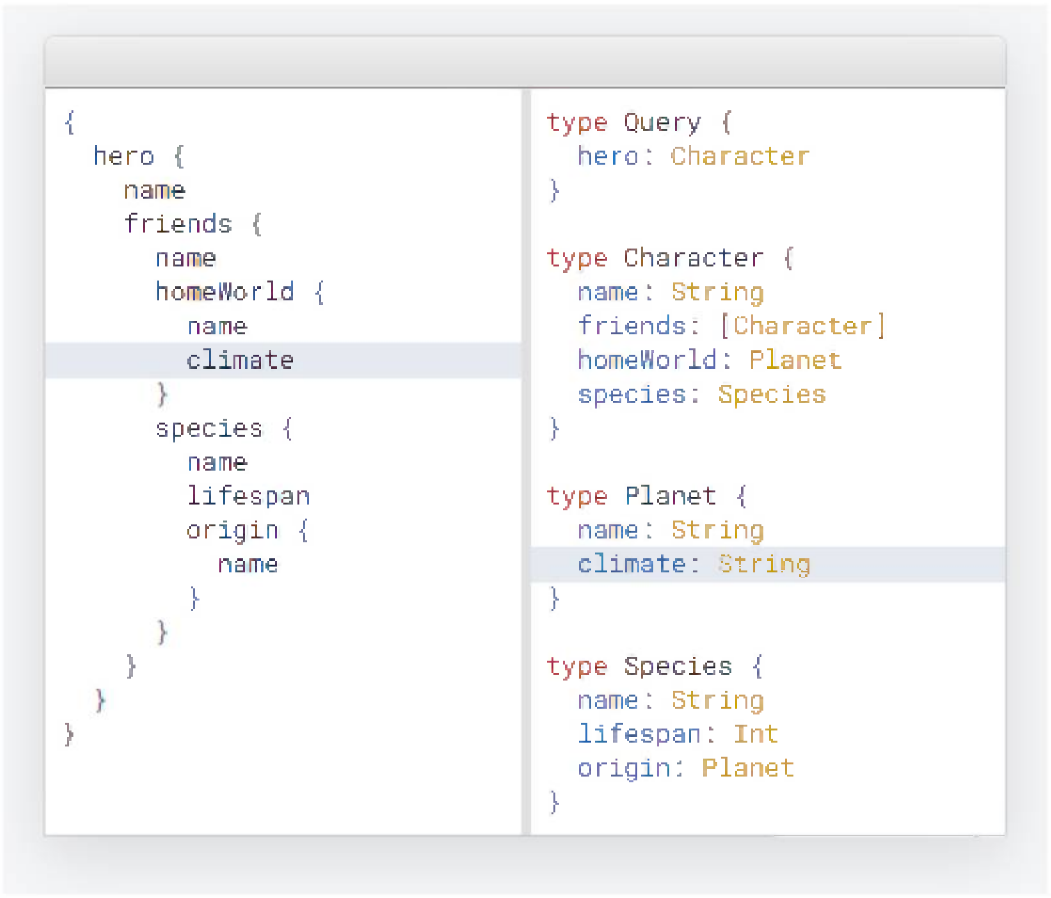
GraphQL

GraphQL is something that both front-end and back-end developers can use. It is a data query and manipulation language for APIs. So you create a GraphQL server on the back-end to work with existing data. It can be used with multiple languages but is pretty popular in the JavaScript/Node.js world.

GraphQL is used in a similar way to a REST API, but there are some advantages. The biggest being that you can **request specific data from a single endpoint**, rather than getting all of it at once like you would with a REST API. This makes applications that use GraphQL fast and stable because they control the data that they receive.

GraphQL is also **strongly-typed** and has strict document requirements.

Take a look at the image below...



```
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
      }
      species {
        name
        lifespan
        origin {
          name
        }
      }
    }
  }
}
```

```
type Query {
  hero: Character
}

type Character {
  name: String
  friends: [Character]
  homeWorld: Planet
  species: Species
}

type Planet {
  name: String
  climate: String
}

type Species {
  name: String
  lifespan: Int
  origin: Planet
}
```

On the left is how we would write a **query**. We are asking for specific fields like the hero's name, friends names, species, etc. On the right is the model which uses **types**. If we want to add or change data, we create something called a **mutation**.

With a REST API, we hit an endpoint and that's that. We get everything that is available no matter what. With GraphQL, we would get exactly what we ask for.

There is a tool called GraphiQL that you can use to make requests and mess around with to learn more about how it works.

I will say that in my own opinion, GraphQL isn't seeming to really take off like many people said it would. I'm not saying it's not popular or telling you not to use it, but there was a point where developers were talking like REST APIs were going to be extinct in favor of GraphQL. I think it's just another example of hype.

Apollo

Apollo is a GraphQL client that you can use to request data. There's all types of packages so that you can use it with different types of front ends. There are other clients as well but Apollo is the most common.

To learn more visit https://www.apollographql.com
To learn more about GraphQL, visit https://graphql.org

Authentication & Authorization

Authentication is the process of identifying an individual and **authorization pertains to the permissions that that individual has**. There are all kinds of ways to authenticate, so I'm just going to give you a very brief rundown of each one. There are a ton of articles and videos that will take you more in-depth.

JSON Web Tokens

When it comes to single-page applications, running on the client, JSON Web Tokens or JWT are a common way of authenticating users. The user validates their username or email and password then a token is signed on the server and sent back to the client. That token can include the user state. The user then has to send that exact token in the HTTP headers when trying to access a protected route. If they don't or the token is a mismatch, they can't access the resource. Visit https://jwt.io to learn more.

Cookies & Sessions

Cookies and sessions are also commonly used in authentication. Just like with tokens, the user data is validated and if successful, the server will create a session in the database and include a *Set-Cookie* header on the response which will have a unique session ID in the cookie object. The browser stores that cookie locally and then it gets sent to the server on each request. The server compares the session ID in the cookie with the one in the database. I do want to mention that there are vulnerabilities to most authentication methods and there are steps to take to minimize them.

OAuth

OAuth is another popular way to validate users and can be a bit easier to set up. It's an open standard for access delegation. So basically, you can log in to a website or an app with your Google account or your Twitter, Facebook, GitHub and many other accounts without having to type in your password. This usually involves setting up some kind of key from within your Google console or Github settings. It's also typical for apps to use both a local method like tokens or sessions along with OAuth and give the user multiple options.

Authentication Libraries

There are libraries to help you implement authentication as well so that you don't have to do it from scratch. *Passport* for Node.js is a common one and lets you use all of the methods above and more.

Password Hashing

It should be obvious that you never store plain text passwords in a database, so password hashing is something that you'll get familiar with. You can do this from scratch, but I would suggest using a library. *Bcrypt* is a tool that I use quite a bit.

Protecting Endpoints & Routes

So like I said, when you use tokens or cookies and sessions, you need to send these with your HTTP request to your API. So you learn to build your API in a way where you protect certain routes. For instance, if you have a route to creating a blog post, you don't want just anyone to be able to open up any HTTP client and create a post. They first have to log in and then send the correct information, otherwise, they should get a status of 403, which means unauthorized.

Two-Factor Authentication (2FA)

Sometimes just having a username and a password is not enough. Especially for applications that store very sensitive data and operations. Two-factor authentication is a method in which a user is granted access after providing two or more (Multi-Factor) pieces of evidence to an authentication mechanism. For instance, in addition to a password, they may have to respond with an SMS code or use an authenticator app, etc.

There are libraries that can help you build a 2FA system including one that I have used a couple of times for Node.js called **Speakeasy**. I believe I even have a tutorial on my channel for it. There are others for other languages as well.

Deploying Full Stack Projects & DevOps

Deploying front-end projects is fairly simple. We have platforms like Netlify and Vercel that make things super easy. We can even create serverless functions to run server-side tasks. When we're talking about a full-blown full-stack application or even an API, hosting gets a bit more complex. As a developer, you are not usually expected to know everything about DevOps. There are DevOps positions for that, however, you should at least know the basics and be able to deploy your projects.

Hosting Platforms

There are a lot of hosting platforms available of all kinds. You have some that set everything up for you as far as operating systems, users, firewalls, etc and some that just give you a Linux container and you install and set everything up yourself.

AWS

Amazon Web Services (AWS) provides on-demand cloud computing to businesses and individuals. As with most of these types of services, they are “pay as you go”. AWS is a very popular option for large projects. Many of the largest websites and platforms in the world use AWS.

There are different services that AWS offers for hosting, databases, storage containers and anything else you can think of. I personally think it’s a bit overkill for single developer projects or even a small team. That’s just my opinion though and I don’t have a ton of experience with AWS, so take that with a grain of salt.

Heroku

Heroku is a platform that I like because it’s relatively easy to use and you don’t have to set up everything from scratch. You simply push your code using Git and it runs on something called a Dyno. There is a Heroku CLI to help you deploy as well as view logs to debug and find issues with your deployment.

You can host Node.js, Python, Ruby and a bunch of other languages. There's an ecosystem of plugins or add-ons as well. It is "pay as you go", but they offer a lot absolutely free as well.

Digital Ocean

Digital Ocean offers cloud servers called "Droplets". They're basically just Linux servers that you can install whatever you want on. This does take some knowledge of Linux and using the terminal. There are other companies that offer similar services such as Linode, which is another company that I've had good experiences with. Both Digital Ocean and Linode have great documentation, so finding help in setting up your server is fairly easy.

Server Software

There are different types of software that you will run on your server

Apache & Nginx

Of course, you do need a web server to "serve" your projects. There are two very popular web servers, Apache and Nginx. They are both open source and used in similar ways. The main difference is in their design architecture. Apache uses a process-based approach and creates a new thread for every request and Nginx uses an event-based architecture to handle multiple requests in one thread.

Apache is very popular in the PHP world as it is part of the **LAMP stack**. Nginx seems to be a bit more popular overall. Nginx is also a bit easier to set up in my opinion.

If you do go with something like Digital Ocean or Linode, you'll need to know how to set up a web server, meaning you'll install Apache or Nginx and configure them. Platforms like Heroku, Netlify and CPanel managed hosting do this for you.

Containers & Virtualization

Containers and virtualization are also very popular, especially in the big business world and among people that plan on getting into developer operations (DevOps)

Docker

Docker is by far the most popular container software. Docker uses operating-system-level virtualization to deliver software packages called *containers*. Containers are isolated from one another and bundle their own software including operating systems, web servers, databases and more.

You could install anything in a Docker container and it runs as its own process on your machine. This is really helpful for teams that work in different environments because the container has the same software and same versions, so your projects will run the same on any machine, whether it’s Windows, Mac or Linux.

Kubernetes

Kubernetes is a container orchestration system for automating development, scaling and management. It’s popular to use Docker and Kubernetes together. Unless you're in DevOps, I really don’t think it's mandatory for you to learn this stuff. It is very powerful and can be very useful for large teams.

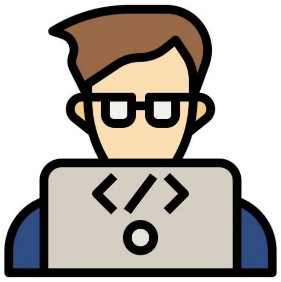
Vagrant

Vagrant is another tool for virtualization. It is used for building portable environments much like Docker, however, Vagrant works with virtual machines and Docker works with containers. Containers are typically smaller and faster.

Image & Video

When it comes to image and video upload, there are a lot of services that you can use. One of my favorites is **Cloudinary**, which doesn't just store your images, but it optimizes them and gives you an API with different image sizes. It can be used with just about anything. Many front-end developers use this type of service as well.

Amazon S3 is a storage service that is part of Amazon Web Services. It provides object storage through a web service interface. It's extremely scalable as well.



Back-end & Full Stack Web Developer

Alright, so to wrap this section up, here are some things that you'll need to be able to do to be a back-end or full-stack web developer.

✓ Programming Language

You should be very comfortable with some kind of programming language that runs on the server. We talked about 9 or 10 of them, so it's up to you on what you want to use. Once you learn a language, try and master it. Don't skip around and learn other languages just for the hell of it. This will just waste time and confuse you.

✓ Database & ORM / ODM

You should understand how to implement and manage a database system. This could be a relational database such as Postgres or MySQL, in which you will also need to learn SQL or Structured Query Language. You may choose to learn and use a NoSQL database such as MongoDB.

Depending on the database and language you learn, you will also need to learn some kind of ORM/ODM such as Mongoose, Sequelize or Eloquent to interact with your database in your software.

✓ RESTful APIs & Authentication

As a back-end or full-stack web developer, you should definitely know how to create a REST API from the ground up including authentication and protecting your routes.

✓ The Terminal

You should also be pretty familiar with the terminal and understand basic Unix commands. You should know how to navigate your system as well as log into remote servers.

✓ Deployment

You should know how to deploy apps and APIs to some kind of platform.

Of course, if you're a full-stack dev, you should be able to do this as well as create user interfaces and the stuff we talked about earlier.

CHAPTER 5

BEYOND THE BROWSER

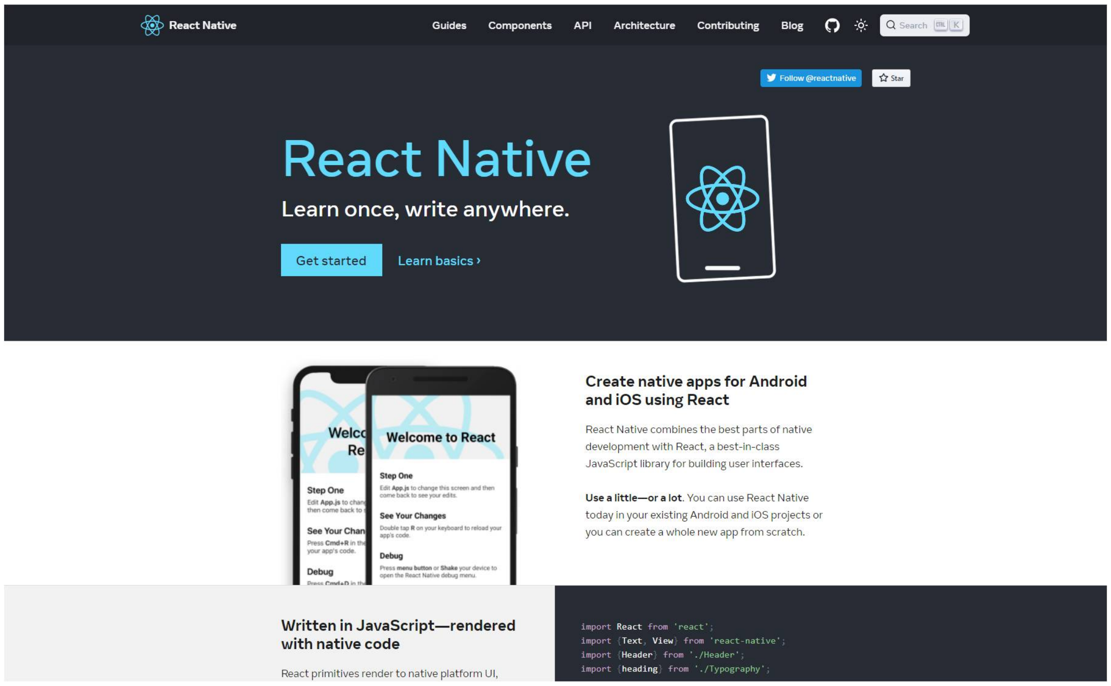
In this chapter we will look at tools to build other types of software such as desktop and mobile apps, with the use of web dev technologies.

Mobile App Development

Mobile app development can be done in quite a few ways with different technologies. You can build native iOS apps by learning the **Swift** programming language and native Android apps using **Kotlin** or **Java,** however, there are some other “web dev friendly” tools to build cross-platform mobile apps.

React Native

React Native is a framework that uses React to build apps for iOS and Android among other platforms with native capabilities. If you’re already a React developer, this is probably going to be your choice. It uses React and the React Native “bridge” that invokes the native APIs for iOS and Android. It essentially uses the same exact React framework as a web app, but instead of using React DOM, it uses React Native.



React Native

Learn once, write anywhere.

Create native apps for Android and iOS using React

React Native combines the best parts of native development with React, a best-in-class JavaScript library for building user interfaces.

Use a little—or a lot. You can use React Native today in your existing Android and iOS projects or you can create a whole new app from scratch.

Written in JavaScript—rendered with native code

React primitives render to native platform UI,

```
import React from 'react';
import {Text, View} from 'react-native';
import {Header} from './Header';
import {heading} from './Typography';
```

Visit https://reactnative.dev for more info

What's great about not only React Native, but most of these web-based frameworks, is that you build once and your app works on both iOS and Android, where you were to build a completely native mobile app, you would have to use different languages for each platform. This saves a ton of time and money.

Flutter

Flutter is a very popular mobile app framework by Google that is used to compile native apps. Flutter uses the **Dart** programming language, which is pretty similar to JavaScript, so a lot of web developers are comfortable with it. Flutter is extremely fast. It's faster than React Native because React Native leverages JavaScript to connect to native components via a bridge. Flutter doesn't need that bridge, which makes it faster. Flutter also has really good documentation.

Ionic

Ionic is a hybrid framework that is great for building progressive web apps or web apps optimized for mobile devices. It is a little slower than the previous two, but you're also not bound to any single framework. If you want to use Angular or Vue or Vanilla JavaScript, you can. Ionic has some great-looking pre-designed components as well.

Xamarin

I wanted to include Xamarin here, which isn't something that I've ever used, but it's a framework for building mobile apps with C# and .NET. It's high performance, cross-platform and the people that I've talked to that use it, seem to really like it. So something you may want to check out if you're coming from the .NET world.

Progressive Web Apps (PWAs)

Progressive web apps are web applications that are built with web development technology and progressive enhancement to give users an experience on par with native apps.

PWA Traits & Technologies

Responsiveness

All PWAs must be completely responsive and must look and feel like a native app on all devices including computers, tablets, smartphones and even TVs. Media queries and viewport are commonly used to achieve this.

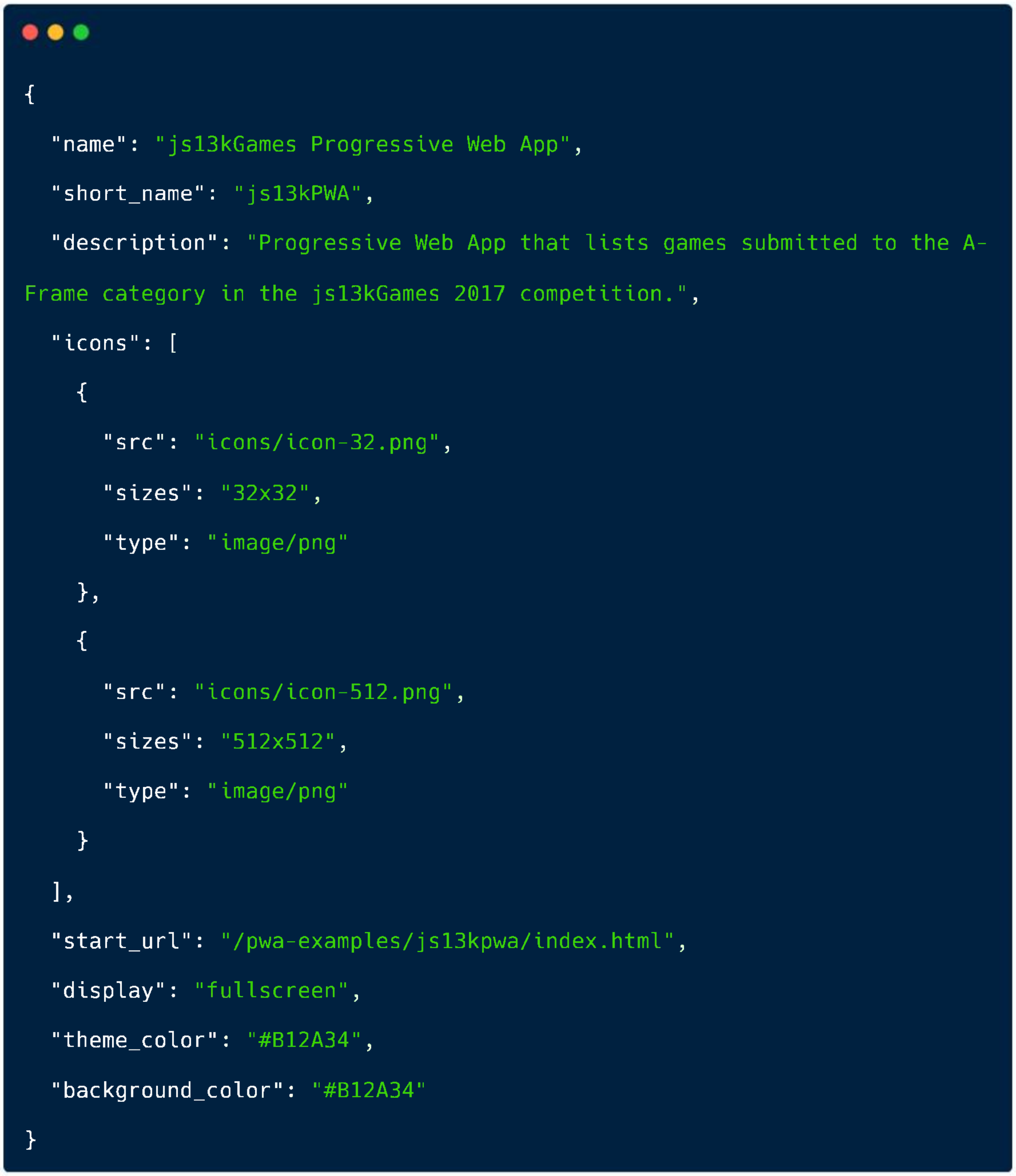
Security

All PWAs must use HTTPS and be developed with security in mind. It should be easy for users to be sure that they are installing the correct app. This is because the users can verify that the app’s URL matches your domain. With app stores, sometimes it isn’t clear. There are many apps that look and sound the same. I’m sure many of you have installed the wrong app at some point. I know I have.

Installable / Manifests

Modern PWAs can be installed on a user's home screen on different devices. This is done with the help of a manifest. A web manifest is a JSON file that provides information about a web app. This is necessary for the app to be downloaded. It usually includes things like the name, author, version, icons and description.

On the next page is an example of what a web manifest may look like.



```json
{
  "name": "js13kGames Progressive Web App",
  "short_name": "js13kPWA",
  "description": "Progressive Web App that lists games submitted to the A-Frame category in the js13kGames 2017 competition.",
  "icons": [
    {
      "src": "icons/icon-32.png",
      "sizes": "32x32",
      "type": "image/png"
    },
    {
      "src": "icons/icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": "/pwa-examples/js13kpwa/index.html",
  "display": "fullscreen",
  "theme_color": "#B12A34",
  "background_color": "#B12A34"
}
```

Offline Browsing / Service Workers

Service Workers are a type of web worker or proxy that sits between the app, browser and network. It is basically a JavaScript file that runs outside of the main browser thread that can intercept HTTP requests, cache data and deliver push messages. They are used so that the web app still has some sort of functionality even when offline or when the user loses their connection. This makes it feel more like a native app, rather than just getting a “website offline” message.

Web Share Target API

The Web Share Target API allows your web app to register in the same way that platform-specific apps can to be able to receive shared content. You will have to update your web manifest to use it.

```
"share_target": {
  "action": "/share-target/",
  "method": "GET",
  "params": {
    "title": "title",
    "text": "text",
    "url": "url"
  }
}
```

Desktop App Development

In addition to mobile development, we now have web developer friendly technologies to build desktop GUI applications.

Electron

Electron is huge in the JavaScript world. It is an open-source framework used to build cross-platform desktop applications with JavaScript. It takes care of the hard parts and lets you focus on your application's functionality. Before you write it off as being non-professional or whatever, let’s take a look at some applications that were built using Electron

■ **VS Code** - Yes the text editor that most of you use was created with Electron

■ **Atom** - Another code editor

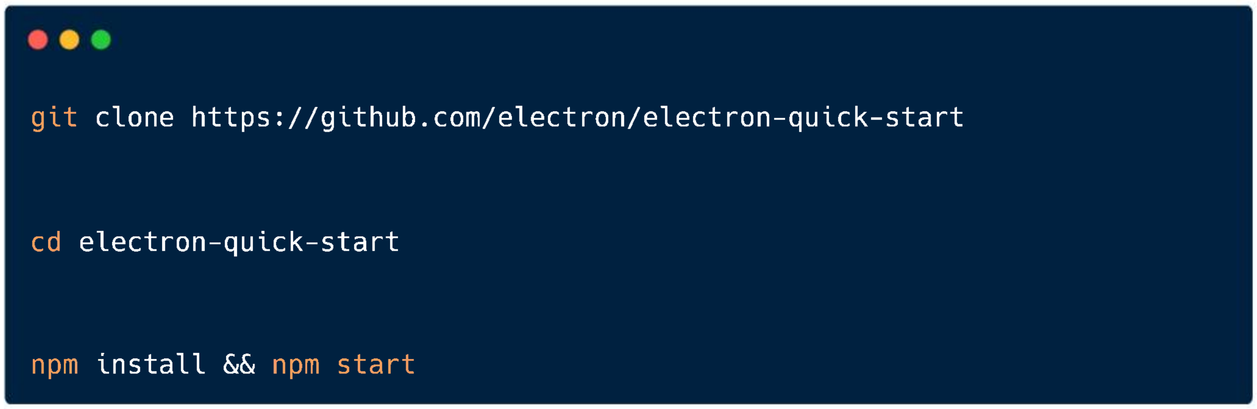
■ **Slack & Discord** - Two giants in the world of online chat messaging

⬛ **Skype** - Popular video chat app

⬛ **OBS** - Live streaming program

⬛ **Postman** - Very popular HTTP client

Getting started is as easy as cloning the quick start repo



```
git clone https://github.com/electron/electron-quick-start

cd electron-quick-start

npm install && npm start
```

Other Options

Some other options for building desktop applications:

✔ NW.js

✔ Python TKinter

✔ PHP Desktop

✔ Native Apps with languages like Java, C#, etc

CHAPTER 6

TRENDS & NEW TECH

In this chapter we will look at some of the newer and popular technology for 2022 - 2023 that may interest web developers.

Web3

Now I want to talk about some stuff that is beyond the traditional scope of a full stack web developer and some other technologies that you might take an interest in.

Web3 has a lot of hype right now. I do think Web3 will be huge. However, I do think that there is a little too much hype right now. With that said, it is good to get a leg up on new technologies. My goal here isn’t to explain in depth what Web3 is, but I want to give a brief overview so that you at least have some idea and you know what you have to start learning to get your foot in the door.

Web 1.0 was basically the “informational web”. Websites, for the most part, were read-only. This was before social media and content creation.

Web 2.0 is the “interactive web”, where the public creates content, however, everything is centralized. Whether you’re on Twitter, Amazon, etc, all the data and control is centralized. W

Web 3.0 or Web3 is the “decentralized web” and uses the Blockchain, which I’ll talk about next. We build something called decentralized apps or **DAPPS**.

Blockchain

The blockchain is basically a **decentralized, distributed public ledger that is immutable**, meaning that it can’t be changed. So think of an actual chain of blocks containing data and each block in the chain contains a number of transactions, and every time that there is a new transaction, it gets added to the ledger.

Nobody can go in and change the data and it's completely decentralized. Unlike centralized servers that can go down, blockchains run on thousands of nodes. The first major use of blockchain technology was Bitcoin, which is a form of cryptocurrency. It’s essentially a digital asset.

Main Advantages of Web3

■ **Ownership of Information or Data:** The end-user will regain the complete ownership of their encrypted data, eliminating the middleman

■ **Privacy:** You control your online identity

■ **Security:** Tamper-proof blockchain technology

■ **Transparency**

■ **Reliability & Uninterrupted Services**

dApps

dApps or DAPPS stands for "Decentralized Applications". The concept is still in its infancy, just like the rest of Web3, so you may see multiple definitions. Some common features that all of these definitions include are the following:

Open Source - All code can be seen and scrutinized by the public

Decentralized - Stored on a public and decentralized blockchain

Incentivized - dApps are typically designed to give users incentive, typically through token rewards. This allows for transparency and without the need for a central authority.

Reliability - Since dApps run on a blockchain, they have no centralized server where if goes out, the app goes offline. If a node drops, another just picks it right up.

It's impossible to take down an application without taking down every other node on the network. If you create a banking application that works better than your actual bank. The bank can not just buy and acquire that application like a normal asset. The dApp will always live on the blockchain.

Many businesses today are built on centralization. Think of the App Store for instance and how much authority they have over your app. This entire way of thinking goes out the window with dApps. They create a much more open, transparent, uncensored and equal web and they take out the middleman.

Ethereum essentially gives us a platform for creating dApps. Let's talk about that next.

Ethereum & Smart Contracts

Ethereum came out a few years ago and is extremely popular because in addition to being a digital asset (like Bitcoin), and maintaining a decentralized payment network, it’s also used for storing code that can be used to power highly secure and transparent decentralized contracts and applications.

Smart contracts are the fundamental building blocks of Ethereum applications. So **if you want to start getting into Web3 and Blockchain, start learning how to create Ethereum smart contracts**.

Smart contracts are essentially “digital contracts” and are stored in the blockchain. Think of a project like Kickstarter, where supporters pay to fund a project and the project owner gets that money when the goal is reached. Kickstarter is the **centralized** entity that controls that money. Both parties have to trust Kickstarter.

With a smart contract, we can get rid of the middleman (Kickstarter) and depend on a smart contract. We don’t need to “trust” a smart contract because they are stored on the blockchain, which is completely immutable, distributed and transparent. It can not be changed once it is created and it is validated by everyone on the network. Crowdfunding is just one example. There are many other types of smart contracts and dApps that you could create.

Solidity Programming

The way that you get into creating Ethereum Smart Contracts is by learning Solidity. Solidity is a statically-typed programming language that was designed to develop smart contracts that run on the Ethereum Virtual Machine or “EVM”. Solidity is a pretty high-level language and if you already know JavaScript or Python, you should be able to pick it up quickly.

NFTs

I should also mention NFTs or “Non Fungible Tokens”. These are basically one-of-a-kind digital assets that are stored on a blockchain. So they can be verified and they can’t be reproduced. The NFT can be artwork, music or anything else. So that’s another part of Web3.

What I don’t like is that there seems to be a lot of scamming going on in the crypto and NFT world. Pump and dump, etc. Hopefully, we can get that under control. But I do expect Web3 and decentralized apps (DAPPs) to be huge in the future.

WebAssembly (WASM)

WebAssembly is another powerful technology that I think we'll see more and more of. It's a low-level assembly language for the web. The purpose of it is to be able to create really high-performance web applications. You can write programs in a very fast language like Rust, C, and C++ and then compile them down to WebAssembly to run in the browser. This allows us to **create web apps that are much more performant than JavaScript** can give us. For instance, programs that are CPU heavy, video processing, high-performance 3D gaming, etc.

If you're a JavaScript developer, there is a language called AssemblyScript which is a variant of TypeScript that can compile to WASM. I should also mention that **WASM is NOT a replacement for JavaScript.** It works alongside it and can be used for the stuff that JavaScript can not be used for.

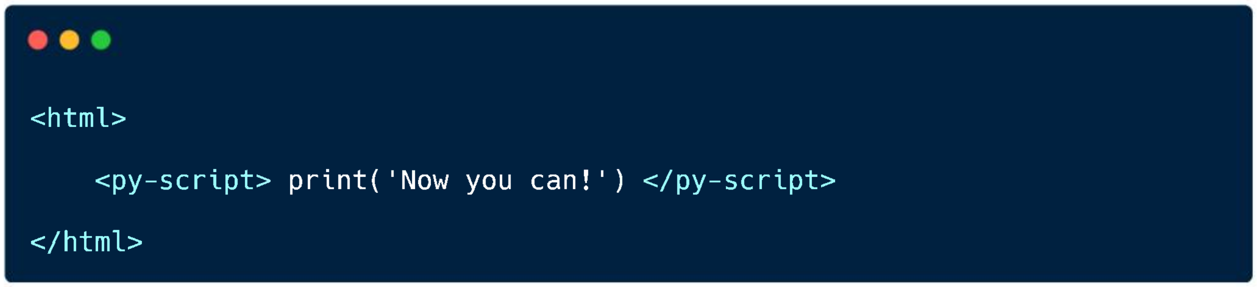
To learn more, visit https://webassembly.org

PyScript

PyScript is a framework that allows users to create Python apps in the web browser. It is something that I have been hearing a lot of lately. It's one of those hype trains on Twitter. In fact, I am working on a video for it now. I usually don't ride the hype train, but I am genuinely interested. It is in alpha right now, so there is a long way to go and you would never use this in production at this time (May 2022).

PyScript uses HTML's interface and the power of **Pyodide** and **Web Assembly**. That's why I wanted to add it right after the WASM section. Pyodide is a port of CPython to Web Assembly. Pyodide makes it possible to install and run Python packages in the browser with something called **Micropip**.

You can use <py-script> tags in your HTML like this



```
<html>
    <py-script> print('Now you can!') </py-script>
</html>
```

To learn more, visit https://pyscript.net

Artificial Intelligence

I think machine learning and artificial intelligence have a role or a couple of roles in web development. We all saw Github Copilot in action in 2021. I use it all the time and it amazes me how well it works. I use it all of the time and it has helped me in many situations. It truly is like a co-pilot. I know a lot of people think it's going to take our jobs, but I don't think that's happening for a long time.

Since AI can mimic human behavior, we are starting to see a lot of it in application testing as well. It makes testing much faster and more efficient. AI-powered tools can be leveraged to solve persistent problems.

There are also machine learning APIs. Amazon has a variety of ML solutions. Translate, Polly and so on. I have not used these personally, but I have read about them and they look promising.

We see AI being used in things like chatbots, spam filters, product suggestions, etc. Machine learning is also heavily used in user engagement and analytics.

Dark Mode

This is kind of a design trend, but many websites and applications are putting in a dark mode setting. It does involve a little bit of JavaScript if you want to add some kind of dark mode switcher. Tailwind CSS actually implements a dark mode helper. You can make certain classes take effect if dark mode is enabled. It is definitely something to think about if you are building websites and user interfaces.

Voice Search Optimization

Voice recognition is super popular right now. We have all kinds of AIs with voice recognition. Web developers are looking into implementing similar features, especially for people with hearing and sight impairments. We talked a little bit about the Web Speech API earlier. This is something that you may want to look into in the near future.

A lot of what we have talked about are also new trends and tech, such as server-side generated pages, PWAs, etc.

VR Devices

VR is only getting better as the years move on. With the popularity of the Oculus Rift and a lot of other devices in the market and coming soon, It is only a matter of time before VR websites are a huge thing.

The **WebVR API** is the central JavaScript API for capturing information on VR devices connected to your computer. With this API, you can get the following types of information from the headset.

✓ Position
✓ Orientation
✓ Velocity
✓ Acceleration

It works by sending stereoscopic images to both lenses of the headset and the position is handled by HMDVRDevice (head-mounted display virtual reality device)

Current browser support is extremely experimental, but it works right now in Firefox nightly builds and experimental builds of Chrome, so you can start playing with it now. Read more here.

CHAPTER 7

OTHER PROGRAMMING CONCEPTS

In this chapter we will talk more about general concepts as opposed to specific technologies.

We already talked about a bunch of programming concepts such as OOP and MVC, but in this chapter, I want to talk about a few other things to think about as a software developer.

The Importance of Fundamentals

I have been coding and teaching for a while and time and time again, I see people jump into frameworks and other things really fast and not only that, but they pay more attention to the framework and remembering syntax than what is actually happening. Obviously, when you start learning any language, you start with fundamentals, but it shouldn’t stop just because you moved on to a framework.

There are different meanings for the word “fundamentals”. There are low-level computer programming and computer science fundamentals as well as programming language fundamentals.

The types of programming language fundamentals that you should be comfortable with are things like the following.

■ Data Types, Conversion & Coercion

■ Functions

■ Control Structures / Conditionals & Iteration

■ Data Flow

■ Input & Output

■ Arithmetic & Logical Operators

This is stuff where if you truly understand how it all works, you can easily pick up another language as a skill. The only thing you will really have to learn is syntax to be able to write in a new language.

There are also fundamentals when it comes to how languages work in general. This is not as important for web developers, because we work at a very high-level and away from the CPU, but it can't hurt to learn this stuff. I did a video on my channel called The Programming Language Guide, which teaches you about the different levels of certain languages, compiled vs interpreted and stuff like that.

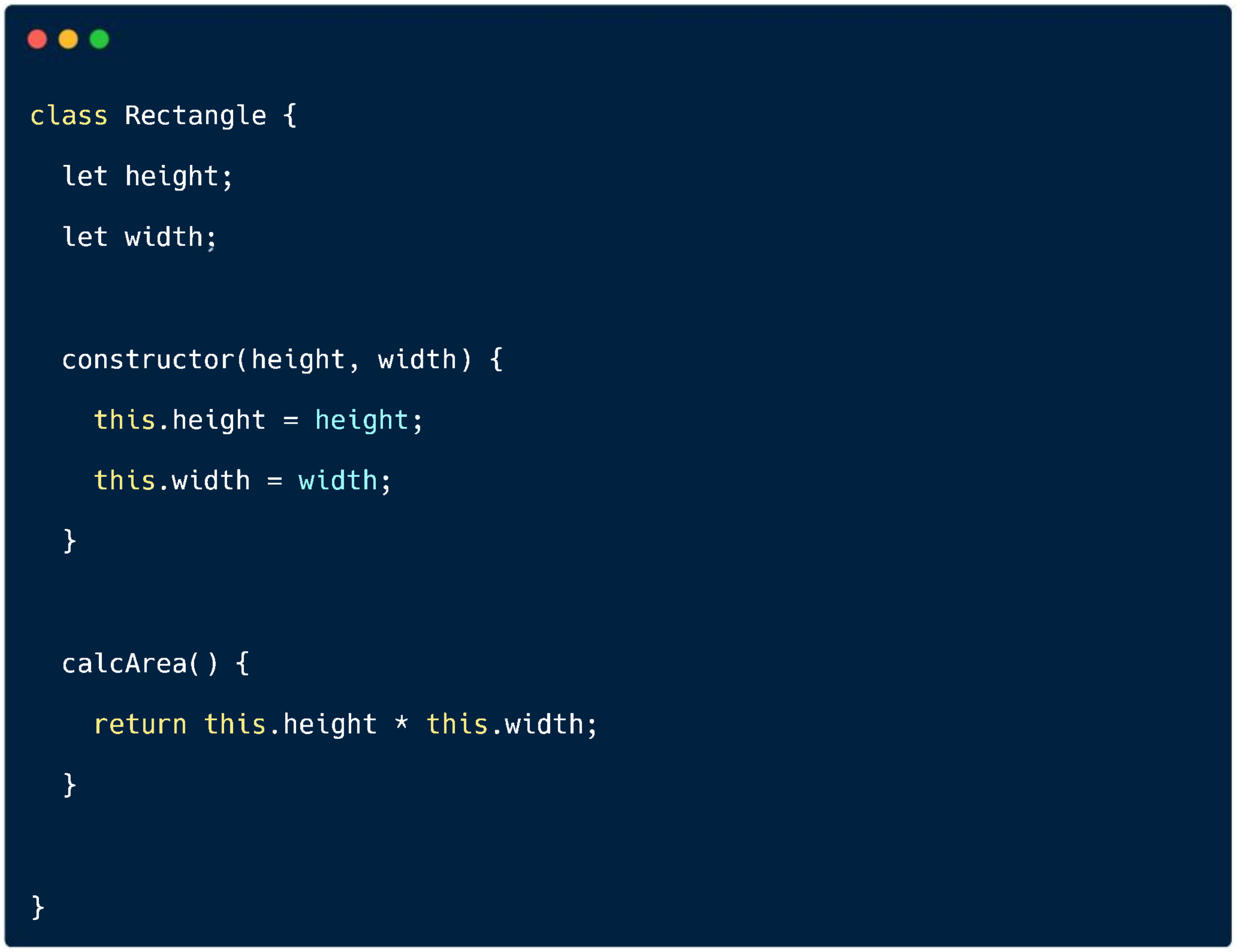
Object Oriented Programming

OOP or Object Oriented Programming is a programming model that organizes software design around data and objects rather than functions. This is something that you definitely need to know especially as a back-end or full-stack developer. A lot of the languages that we looked at are OOP or at least have the capability of being OOP and many of the frameworks that I mentioned use it in some form or another.

To explain OOP in simple terms, we look at everything as an object. So a user, a blog post, any kind of resource is an object. These objects are created or **instantiated** using **classes.** So you have an **instance** of an object. In your classes, you can have **properties**, which are just variables or attributes and you can have **methods,** which are just functions of that class. A **constructor** is a special kind of method that runs when you create a new object.

The way that you create objects is different for every language. JavaScript for instance is a **prototype based** object oriented language. It does not natively have classes. It defines behaviors using a constructor function and prototypes. However, with ECMAScript 6 (ES2015), they added the ability to use classes. It is still doing the same thing under the hood, but you have what is called **syntactic sugar,** which is an abstraction of what is really going on. This is so the developer can do things in an easier way.

I will use JavaScript class examples to show how OOP works, however, the process is similar for most languages. The syntax just varies a bit.



```
class Rectangle {
  let height;
  let width;

  constructor(height, width) {
    this.height = height;
    this.width = width;
  }

  calcArea( ) {
    return this.height * this.width;
  }

}
```

Above is a class to create a Rectangle object. It has 2 properties, height and width. Initially, they are not defined, they are just declared.

The constructor is defined, which runs when the object is created. It takes in a height and width value and then sets those values to the properties height/width. To do so, we use the **"this"** keyword. It pertains to the current object.

There is another method called calcArea, which multiples the height and width and returns that value.

Right now, this code does not do anything. It is just a defined class. To create a new object using that class (instantiation), we would do this.

```
const square = new Rectangle(10, 10);
```

To get the area of this square we would call the calcArea() method

```
const area = square.calcArea();
console.log(area)
```

We can also access properties of the square object

```
console.log(square.height)
console.log(square.width)
```

This is a very simple example. To learn more I have a YouTube video called JavaScript OOP, if you are interested.

Encapsulation

You may have heard this word quite a bit. I could write a chapter on encapsulation, but In simple terms, encapsulation in OOP refers to the bundling of data and the methods that operate on the data, into a single unit. Classes are a common form of encapsulation. Encapsulation also refers to the mechanics that restrict certain access to some components. Some benefits to encapsulation are:

✔ Security & Hidden Data
✔ Flexibility
✔ Reusability

Algorithms & Data Structures

Once you learn the fundamentals that we talked about and you are programming at a decent level, you should always be challenging yourself and algorithms and data structures are a great way to keep working out that muscle in your brain.

Algorithms are **procedures or formulas for solving specific problems**. They work by following a procedure or a set of instructions made up of inputs.

I have seen people say that practicing algorithms are useless and not practical. Honestly, I felt the same way at one point. I thought, when are you actually going to do a FizzBuzz or palindromes in the field? Well, I came to realize that you probably won't do that exact same thing, however, you will need to solve problems using specific ideas and methods and that is what you learn, is how to solve problems with your programming language of choice. With JavaScript, algorithms have really taught me more about high order array methods like filter, reduce, etc and I use these all of the time in real life.

As far as learning and challenging yourself with algorithms, there are some popular websites that can help you with that such as

✔ Codewars

✔ Project Euler

✔ Hackerrank

✔ LeetCode

✔ CoderByte

I also have a playlist on my channel called “JavaScript Cardio” with some beginner and intermediate challenges. It is only 3 videos, but I plan on adding more.

Data structures will help you manage and organize data so that you can perform operations in an efficient manner. There are primitive data structures that pretty much everyone knows, like arrays, but you also have structures that you can create as a programmer, for instance,

✔ Linked Lists

✔ Queues

✔ Stacks

✔ Trees

✔ Graphs

✔ Hash Tables

Learning how to create and work with these structures is extremely valuable.

If you are looking to work as a programmer at a large tech company, then you NEED to learn some of this stuff just to get hired. I honestly don't think it's a great way to test someone's skill, however, this is what many companies do. So I would definitely suggest learning some basic stuff like FizzBuzz, anagrams, string reversals, etc to prepare for some of the interviews you will be going through.

Software Design Patterns

A design pattern is a general re-usable solution to commonly occurring problems and a way to structure your code to function efficiently. As you become a bit more advanced as a developer, you will probably at some point, look into different software design patterns. Because of the fact that we have so many frameworks, this is less of a need today, but they do deserve some attention. Especially if you are at the point where you can build your own frameworks.

Let’s take a quick look at some popular design patterns. There is a video on the Traversy Media channel by Jack Herrington about these 5 patterns, if you want to learn more. These are just high-level definitions.

Singleton Pattern

Restricts the instantiation of a class to a **single instance**, hence the name. You may not need to create multiple objects. In fact, this is true for many situations. So the singleton pattern is very useful when you only need one object.

Facade Pattern

Used in OOP. It is an object that acts as a front-facing interface masking more complex underlying code. It is used to improve the readability and usability by masking interaction with complex components behind a single more simplified API.

Bridge/Adapter Pattern

Works as a bridge between incompatible interfaces. It allows the interface of an existing class to be used as another interface.

Strategy Pattern

Behavioral design pattern that enables the selecting of an algorithm at runtime. It encapsulates a group of algorithms and selects one from the pool. These algorithms are interchangeable and one can be substituted for the other. Typically, it stores a reference to some code in a data structure and retrieves it for use.

Observer Pattern

An object, called the **subject**, maintains a list of dependencies, called **observers**. The observers are notified when any part of the state is changed, usually by calling a specific method. This pattern is commonly used in event handling systems.

These are very simplistic general definitions, so if you are interested in learning more, I would advise you to look further into these patterns as well as others.

Clean Code

Keeping your code clean and using best practices are very important. I do think some developers go overboard with this and care too much about things that just do not matter, but there are also developers that are sloppy that don't care at all as long as it works. It is best to be in the middle ground here.

Let's look at some good practices:

Good Naming Conventions

This is one of the most important parts of writing clean code. This pertains to naming variables, functions, classes and anything else. I try to keep these 3 things in mind when naming anything.

✔ Be Consistent

✔ Have Simplicity

✔ Be Descriptive

If you use camel case then always use camel case. If you use underscores, then always use underscores. I understand different objects may have certain conventions. For instance, model names in many languages/frameworks are often pascal case (start with an uppercase letter). That may differ from the normal convention.

Sometimes, simplicity and being descriptive go against each other. As a simple example, you have the option to use either

```
loadConfigFromServer()
```

Or

```
loadConfig()
```

One is very descriptive and one is a bit simple and clean. Well, how obvious is it that the config is being loaded from the server? If it is pretty obvious, I would use the simpler name and maybe add a comment if needed.

```
// Load config from server
loadConfig()
```

Here are some more specific naming rules to follow:

✓ Class names should be a noun phrase (eg. User, UserController, UserModel) and methods should be verbs (eg. login, getPosts)

✓ Pick one word per concept. For instance, don't use fetchPosts() and then somewhere else use getUser(). Pick either fetch or get and stick with it. Again, consistency is very important in programming.

✓ Write code like other people will need to read it, even if that isn't true. This will help you enforce good practices on yourself.

Keep It Simple (KISS)

KISS or **"Keep It Simple"** is a key design principle in programming. Many programmers are the arrogant "know it all" types that like to be competitive when it comes to code. Maybe because they are so bad at sports, I don't know. Just kidding. Programmers can be good at sports too. Unfortunately, this can cause them to do things that are over-complicated. They may be clever or "outside the box", but only they can understand it. I am against this kind of thing. If it makes the code more efficient and it may be worth it, then sure, but to just do it to show how clever you are, is silly.

As I stated above, always write code as if others need to understand it. Even if nobody else will read it, I guarantee that if you do this sort of thing and write over-complicated code, you will go back to it in a year and not remember what the hell you did. I think it's fine to do for fun, but if you are building a serious project, keep it smart and simple.

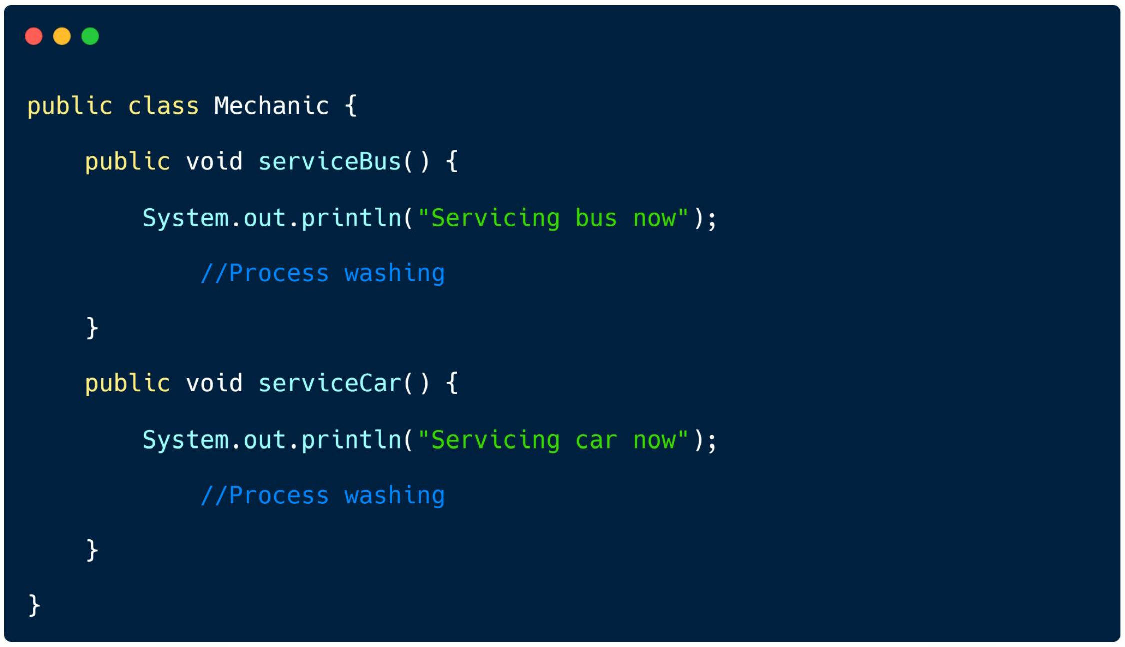
One tough thing about programming is that there are usually 50 ways to do the same thing. This causes a lot of confusion and it can be hard to tell which ways are good and which ways are bad. So coming up with a solution that is simple and easy to read is usually the better way.

Keep It DRY

DRY stands for **Don't Repeat Yourself**, which is another core principle in writing clean code.. One of the biggest mistakes new developers make is writing the same code over and over. Look for patterns in your code and when you see stuff starting to repeat, think about breaking it up into functions or use iteration where necessary. This will not only save you lines of code, but it will make your application more scalable and performant.

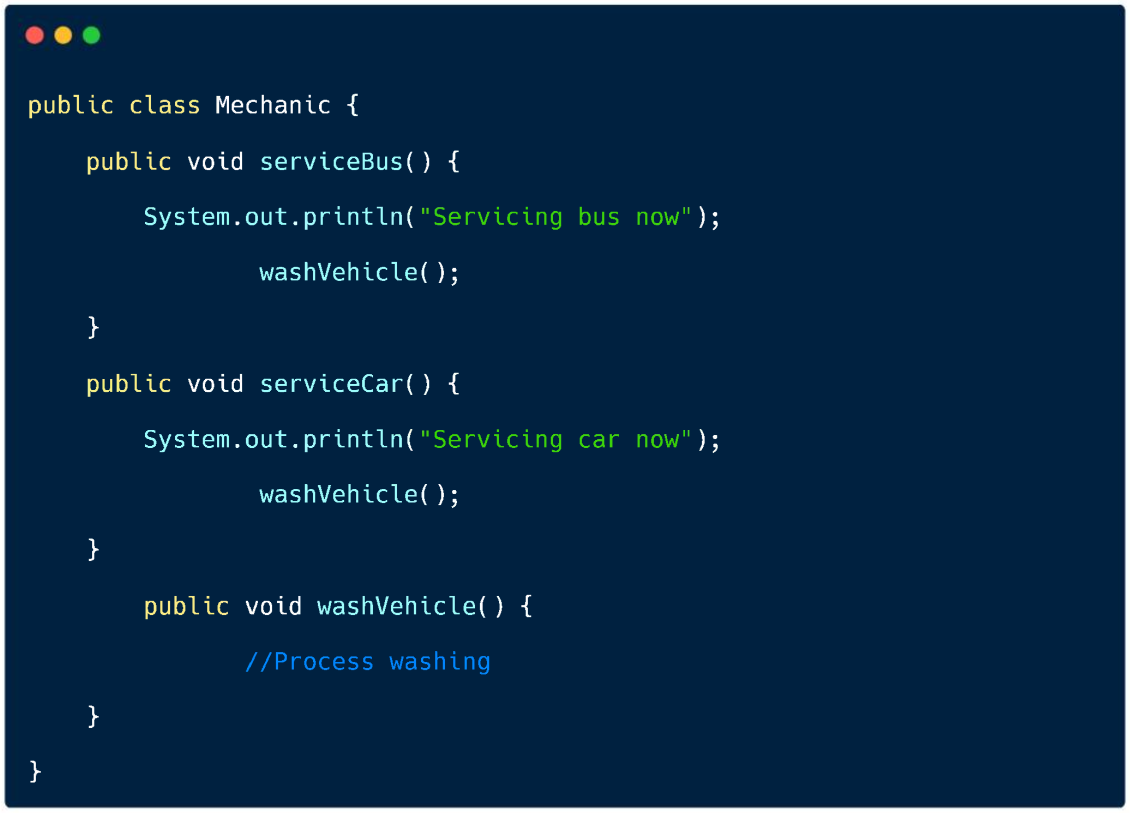
Let's look at a Java example from w3Schools on the next page.

Non-DRY



```java
public class Mechanic {
    public void serviceBus() {
        System.out.println("Servicing bus now");
            //Process washing
    }
    public void serviceCar() {
        System.out.println("Servicing car now");
            //Process washing
    }
}
```

DRY



```java
public class Mechanic {
    public void serviceBus() {
        System.out.println("Servicing bus now");
                washVehicle();
    }
    public void serviceCar() {
        System.out.println("Servicing car now");
                washVehicle();
    }
        public void washVehicle() {
               //Process washing
    }
}
```

Instead of having the same bit of code that does the washing for both vehicles in two or more places, have a single function for washing a vehicle.

It is not always about saving lines of code. This does not even save any lines, but it does make it scalable so that if you decide that you need to use that piece of code again later and add a new vehicle, you can easily reference the function instead of repeating yourself.

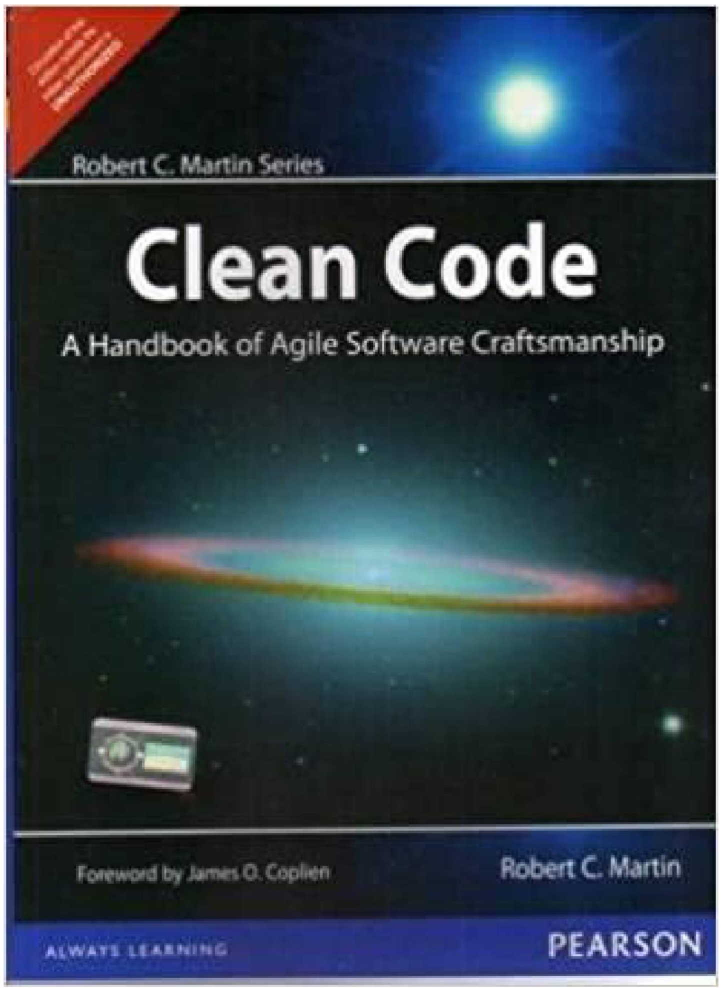
Proper Commenting

Every developer has different opinions on this. Some say you should comment any and everything, some say you don't need to write any comments because the code should be descriptive enough. Again, go for the middleground. I do think that you should write comments to describe things that are not "run of the mill", but be descriptive to avoid having to write too many comments. If you have a function called getUserById(), there really is no need for a comment like this one

```
// This function gets users by their ID
function getUserById() {}
```

This is completely redundant in my opinion. I may do something like this in a course, but I would not in my own production code.

In conclusion, there is a lot that comes down to preference, but there are best practices that you should be following. If you are really interested in learning more about best practices, check out the book **Clean Code** by Robert Martin.



CHAPTER 8

MANAGEMENT & LIFESTYLE

In this chapter we will look at some ways to manage projects as well as lifestyle issues such as physical and mental health.

Project Management

We talked a lot about tools and technologies, so I want to touch on project management. When you actually start building things, whether personal projects or projects for your job or for freelance clients, you are going to need to plan and document the goals and the tasks of the project.

Some people are very organized and have a strict process when it comes to project management and some people are more “off the cuff” and take it as it comes. It is best to be somewhere in the middle.

If you are freelancing or running a business, then you are the one responsible for managing the project from start to finish. You have to put a lot more thought into this stuff than someone that works for a company because most companies already have specific methodologies, software and ways of doing things.

When you start a project, there are a few things to think about and figure out.

✔ Main Goal (Sell a product, collect email addresses, get clients, etc)

✓ The General Timeline

✓ Gather resources (Images and other content)

✓ What you are charging for the work (If it is for a client)

✓ What software & tools you will use

Project Management Software

Every developer manages their projects in different ways. Some do not use any specific software, especially if they work alone. I went years jotting stuff down in a text file. Eventually, I started using project management software for both development and content creation.

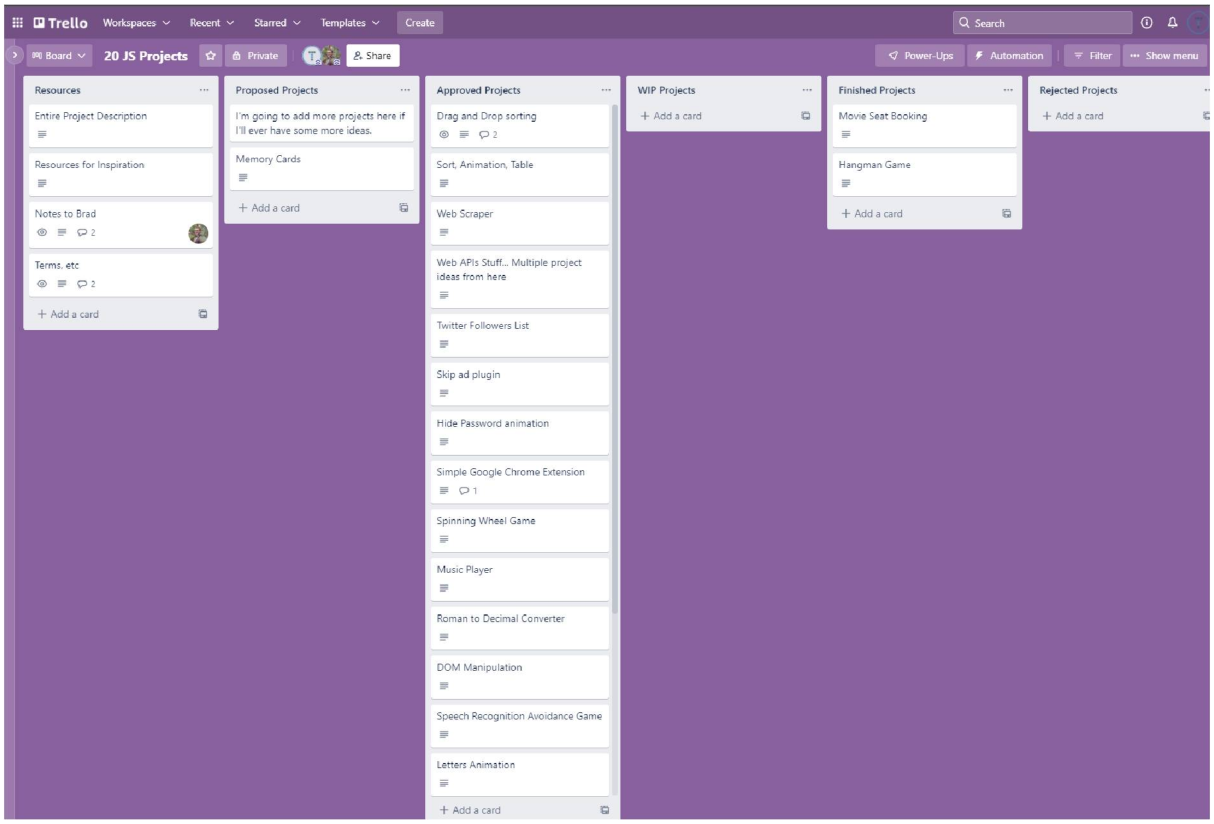
Let's take a look at some popular options:

Notion

Notion is what I use for everything. I love it because you can create any type of project, whether it's a YouTube video or a web app. You can have all kinds of custom fields for endless content types. It is great for people that work alone, but also for teams. There are a lot of collaborative features. You can use it for free, but there are premium accounts as well. You can use it in the browser, mobile app or download the desktop app.

Trello

Trello is another app that I have used quite a bit. You basically create what are called "trello boards" and add different tasks and items to them. If you like very simple software like I do, this is a great option. Add the features that you plan to build and check them off as you go. Easy peasy.



Resources
- Entire Project Description
- Resources for Inspiration
- Notes to Brad
- Terms, etc

Proposed Projects
- I'm going to add more projects here if I'll ever have some more ideas.
- Memory Cards

Approved Projects
- Drag and Drop sorting
- Sort, Animation, Table
- Web Scraper
- Web APIs Stuff... Multiple project ideas from here
- Twitter Followers List
- Skip ad plugin
- Hide Password animation
- Simple Google Chrome Extension
- Spinning Wheel Game
- Music Player
- Roman to Decimal Converter
- DOM Manipulation
- Speech Recognition Avoidance Game
- Letters Animation

WIP Projects

Finished Projects
- Movie Seat Booking
- Hangman Game

Rejected Projects

Zoho Projects

Zoho Projects is a great tool for teams. It allows you to see the big picture and track your scheduled tasks and pay attention to what is critical and what isn't. It has a very in-depth, but intuitive UI with all kinds of themes. You can also integrate other apps like Slack, Google products, etc.

Project.co

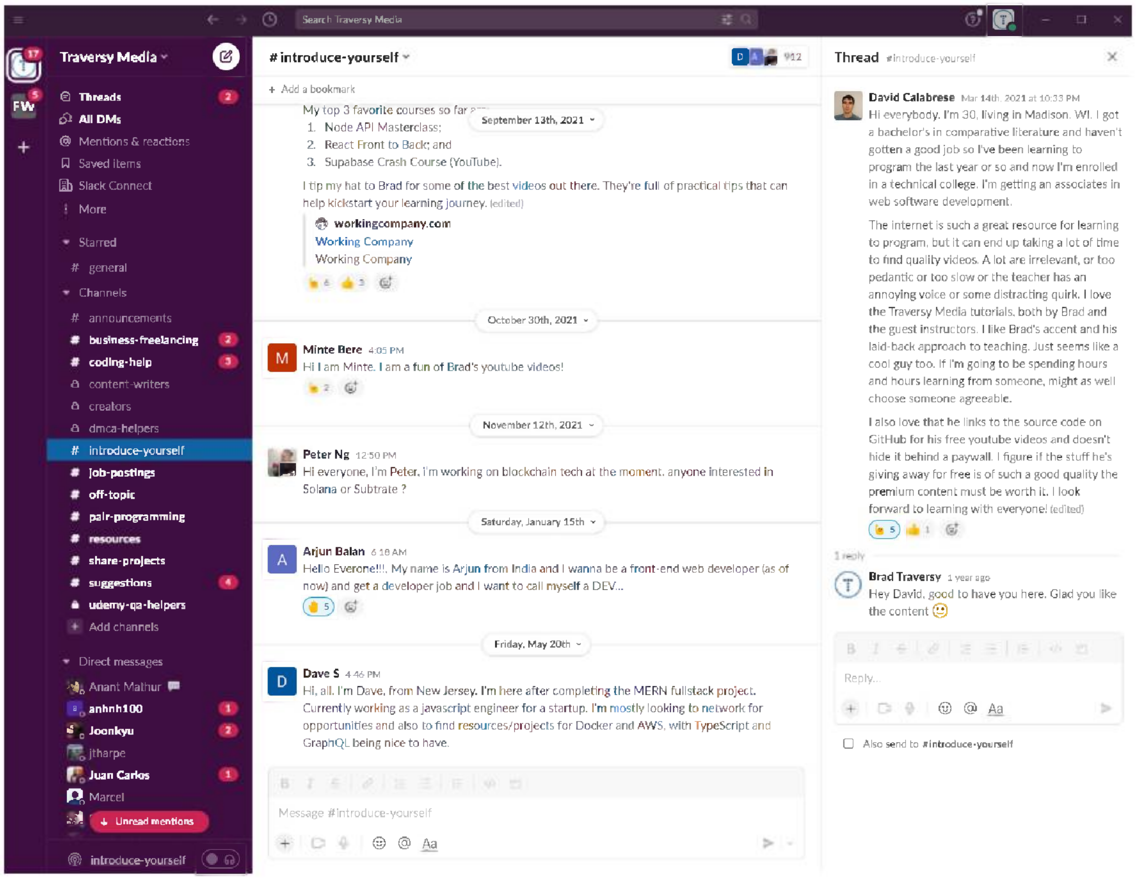
This is something that I recently found that looks like it would have been great when I was taking clients. It isn't just for your own project management, but it also includes the client. They can have discussions with you. You can share resources, track time if you work by the hour and even take payments right through the platform. There is a free option for 1 user and $10 per month for additional users.

Communication Apps

These days a lot of people are working from home. There are many companies hiring for remote work. Obviously, we still need to communicate and collaborate. There are many options, but here are some popular ones.

Slack

Slack is probably the biggest text chat software in the business world. It is owned by Salesforce and offers many IRC-style features. You can create chat rooms organized by topic, private rooms, direct messaging and much more. If you need to share files, it's as simple as dropping the file into the chat or DM.



My top 3 favorite courses so far are:

1. Node API Masterclass;
2. React Front to Back; and
3. Supabase Crash Course (YouTube).

I tip my hat to Brad for some of the best videos out there. They're full of practical tips that can help kickstart your learning journey. (edited)

workingcompany.com
Working Company
Working Company

September 13th, 2021

October 30th, 2021

Minte Bere 4:05 PM
Hi I am Minte. I am a fun of Brad's youtube videos!

November 12th, 2021

Peter Ng 12:50 PM
Hi everyone, I'm Peter. i'm working on blockchain tech at the moment. anyone interested in Solana or Subtrate ?

Saturday, January 15th

Arjun Balan 6:18 AM
Hello Everone!!!, My name is Arjun from India and I wanna be a front-end web developer (as of now) and get a developer job and I want to call myself a DEV...

Friday, May 20th

Dave S 4:46 PM
Hi, all. I'm Dave, from New Jersey. I'm here after completing the MERN fullstack project. Currently working as a javascript engineer for a startup. I'm mostly looking to network for opportunities and also to find resources/projects for Docker and AWS, with TypeScript and GraphQL being nice to have.

David Calabrese Mar 14th, 2021 at 10:33 PM
Hi everybody. I'm 30, living in Madison, WI. I got a bachelor's in comparative literature and haven't gotten a good job so I've been learning to program the last year or so and now I'm enrolled in a technical college. I'm getting an associates in web software development.

The internet is such a great resource for learning to program, but it can end up taking a lot of time to find quality videos. A lot are irrelevant, or too pedantic or too slow or the teacher has an annoying voice or some distracting quirk. I love the Traversy Media tutorials, both by Brad and the guest instructors. I like Brad's accent and his laid-back approach to teaching. Just seems like a cool guy too. If I'm going to be spending hours and hours learning from someone, might as well choose someone agreeable.

I also love that he links to the source code on GitHub for his free youtube videos and doesn't hide it behind a paywall. I figure if the stuff he's giving away for free is of such a good quality the premium content must be worth it. I look forward to learning with everyone! (edited)

1 reply

Brad Traversy 1 year ago
Hey David, good to have you here. Glad you like the content 🙂

The free account gives you access to the last 10,000 messages. There is a pro, business and enterprise level membership as well for more features and storage.

Discord

Discord is also very popular and is very similar to Slack. I would say that Slack seems to be more business oriented and Discord is used for a lot of other things like gaming and other hobby-based chats. It can work well for business too. I use Discord quite a bit for communication with other creators and programming chat. Discord is also great for voice calls. I use Slack for more professional communication, but both are great. I would suggest installing both on your computer.

Discord is free to use, but there is a Discord Nitro feature that gives you an enhanced experience with things like bigger file uploads and different emojis. It is $9.99 per month or $99.99 per year.

Zoom

Discord and Slack are great for text and audio, but if you need to have a video conference with one or more people, then Zoom is a very popular option. During Covid, my kids had to do their school via Zoom and of course there were some technical issues, but we figured it all out and it worked pretty well.

Zoom is free, but like other similar software, there are premium options if you want to host more people, etc. I believe with the largest package, you can host 500 people. Which honestly, seems like a nightmare to me.

Methodologies

A methodology is a system of principles, techniques and procedures used by those who work in discipline. They are often used at large companies. You probably won't be using a methodology if you are freelancing or creating your own projects. Collaboration is a big part of using a methodology.

There are a few project management methodologies. If you work for a company, this will probably be chosen for you. Let's take a quick look at some of the popular ones.

Agile

Agile uses a sprint approach and the project deliverables are separated into “cycles”, and focuses on flexibility and collaboration. Instead of betting everything on a “big bang” launch, they work in small, but consumable increments.

There are 4 general principles of agile.

✔ Individuals and interactions over processes and todos
✔ Working software over comprehensive documentation
✔ Customer collaboration over contract negotiation
✔ Responding to change over following a plan

Agile seems to work well for really large companies with large teams. It is very “people over process” driven and collaboration + communication is key. If you are running a small business, you probably will not be using agile. I personally have never used this methodology.

Waterfall

The Waterfall methodology emphasizes a linear progression from start to finish. It is used by developers as well as engineers. It involves careful planning and detailed documentation. It is much more straightforward than agile and is definitely more my style.

Waterfall works well for projects with long timelines and detailed plans. The project is mapped out into different phases and one phase is only started when the last is completed, working in a linear fashion to complete an end goal. Everyone involved in the project has a very clear role and sticks to that role.

The typical stages of waterfall:

✓ **Requirements** - Manager gathers requirements needed
✓ **System Design** - Design workflow model
✓ **Implementation** - Team begins work
✓ **Testing** - Test each element
✓ **Deployment** - Launch service or product
✓ **Maintenance** - Perform upkeep and maintenance

Scrum

Scrum is a project management framework for developing and delivering products in a complex environment. This is another methodology you will see with large teams at companies like Microsoft and Google. It’s also used in other non-developer worlds like sales and research.

Scrum advocates for planning at the start of a “sprint”. A sprint is a repeatable fixed time-box during which a “done” product of the highest possible value is created. Sprints are used with agile and scrum.

There are 3 groups of phases named pregame, game and postgame

Pregame is all about planning and the project architecture. In this phase, a list of activities and a backlog is created by breaking everything down into smaller “user stories”.

The **game** is where the actual work is done. Each task is assigned, starters and completed until the definition of “done”. This also includes review meetings and making needed changes.

Postgame is where the products are prepared for release. Testing, integration, training and documentation are done in this phase.

There are many other methodologies. If you want to work for a large tech company, you will most likely have to work within parameters like this. Personally, this is not my style. I like doing things my own way, which is why I have been more of an entrepreneur.

I don't work as well with others and it isn't because I think I know best. In fact, I am very sure that I don't. It's because I like to be left alone and do things in a way that works for <u>me</u>. This doesn't mean that methodologies are not for you. Everyone is different.

Physical Health

Many people think that being a developer is a "cushy" job because we sit a lot and don't have to lift heavy objects. I grew up around blue collar workers, so I have always heard this. However, sitting all day, especially if you have bad posture is absolutely horrible for you. It can do more damage than hard labor in some cases. I have messed up my neck from all the years of sitting with bad posture. Here are some common health-related issues that are common for developers.

✔ Back, neck and shoulder issues

✔ Carpal tunnel

✔ Vision problems & dry eyes

✔ Obesity from not enough movement

✔ Bad diets

✔ Sleep issues

✔ Headaches

✔ Sciatica

Tips To Being A Healthy Developer

Here are some tips that I have come across to combat some of these issues, mostly from my own experience.

Take Breaks

Be sure to take short and long breaks. I usually take a 5-10 minute break every 1 - 2 hours. Some people do much shorter intervals like every 30 minutes. I just know that if I am in the zone, I won't stop every 30 minutes. Take 1 or 2 long breaks as well. I take 90 mins around noon to go to the gym.

Exercise

The last sentence above brings me to my next tip, which is to exercise. You don't have to go powerlift every day for 3 hours, but try to work in a short exercise routine. Even if you just go for a walk a few times per week. All developers are still for long periods of time. Make sure you move around.

Get a Sit/Stand Desk

I know they are expensive, but if you can afford it, sit/stand desks are well worth it. Just make sure you actually stand. I split it up around 60% sitting and 40% standing throughout the day. Some days maybe 70/30.

Rest

Be sure to rest. Don't stay up all hours of the night coding. If you overdo it, you will just hurt your productivity. Try and get at the very least, 6 hours of sleep per night, but shoot for 8.

Mental Health

In my opinion, being a developer affects mental health even more than physical health and in so many ways.

Burnout

Burnout is one of the worst things about being a developer. It happens to all of us. Nobody can escape it. I had a very good run. It took me a couple years before my first really bad burnout. I worked like a machine to get to where I am. It took a toll on me though. Now I burn out a couple times per year.

If you are unclear on what burnout is, it is when you push yourself too hard. Maybe you work 7 days per week or 16 hour days or both. When you code, your brain is using a lot of energy. You start to run on fumes and become very foggy, moody and have an overall feeling of depression. If you are like me and you are actually diagnosed with depression, then it is even worse.

The worst thing that you can do is try and work through your burnout. Believe me, I have tried. You only make things worse. The best thing to do is to take a few days off and rest your brain. I know not everyone can do this when they need it. If your job won't let you or you have a client, etc, then do it the first chance that you get. Do it even if you start to feel better because it will come back. Try to relax and do some things you enjoy that don't take much brain power. I like photography and making music. It helps me unwind and clear my head.

Imposter Syndrome

Imposter Syndrome is something that runs rampant in the software development world. If you don't know what Imposter Syndrome is, it is when someone feels like they are undeserving of their achievements and they usually have somewhat low-self esteem about their career. They essentially feel like a fraud. I should mention that Imposter Syndrome is NOT an actual psychiatric disorder. It is just a label that we put on this type of feeling and situation.

I think that a lot goes into this. I think one of the main reasons is because, as you saw from this book, there is SO much to learn and it is impossible for anyone to learn everything. So there is always going to be stuff that you don't know. This can make you feel inadequate and like everyone knows more than you.

Another reason is because on the surface, it seems like everyone else knows exactly what they are doing. The truth is that most other developers feel just like you do. You just don't see their failures and struggles. You only see your own.

This is hard for some people to believe, but I have an incredible amount of Imposter Syndrome, even until this day. I have a programming channel with almost 2 million subscribers and I still wonder if I am made to do this. It occurs a lot less than it used to, but it is definitely still there. In fact, I think the pressure of being in my position can really make it worse, because I am expected to know so much.

It will get better. I can't say that it will go away completely, but it will happen less often as you progress in your career and gain more knowledge. It seems like everyone in tech either doesn't think that they are good enough or they think that they are Gods. Try and be somewhere in between ●

Social Life

In my experience, it seems that a lot of people that are socially awkward or have social anxiety are drawn to this line of work. I'm not sure why, but I am guessing it is the nature of the job. You spend a lot of time by yourself with a computer with little interaction with other human beings. I know developer jobs at large companies can be very collaborative when you work on a team, but not all jobs are lke that and many developers, like myself, work alone.

I would highly recommend trying to get out and spend some time with family and friends. I know not everyone has family and friends, so if you don't, then try and find some social groups or even something like a tech meetup. Just to get some human interaction.

For those of you like me that have a family, be sure to spend as much time with them as possible. I went years working way too much and not spending enough time with my kids. When my little daughter said "Dada, you are always working", that is when it hit me that I'm not going to be on my deathbed saying I wish I did just one more project or got one more client. Work is important, but make sure that you also get out there and live your life!

Wrap Up

I hope you enjoyed this book. I know that this is a lot of information. I think the most important thing to understand is that **you do not have to learn all of this stuff.** My goal was to simply help you understand what each of these technologies are used for and maybe help you choose your path as a developer. I know that It can be difficult just to learn what to learn. Plan and take it one step at a time and you'll be absolutely fine.

Thank you so much for reading this book! I really hope that it helps you on your journey!

Links

Here are the links to all of the technologies mentioned in this guide.

Traversy Media

Traversy Media Website

https://traversymedia.com

Video Guide

https://traversymedia.com/guide

Slimmed Down Video Version Of This Guide

https://www.youtube.com/watch?v=EqzUcMzfV1w

Browsers

Google Chrome

https://www.google.com/chrome/

FireFox

https://www.mozilla.org/en-US/firefox/new/

Safari

https://www.apple.com/safari/

Brave

https://brave.com/

Microsoft Edge

https://www.microsoft.com/en-us/edge

Text Editors

VS Code

https://code.visualstudio.com/

Sublime Text

https://www.sublimetext.com/

Atom

https://atom.io/

Vim

https://www.vim.org/

iTerm2

https://iterm2.com/

Git Bash

https://git-scm.com/downloads

Hyper Terminal

https://www.hilgraeve.com/hyperterminal-trial/

Design Software

Figma

https://www.figma.com/

Adobe

https://www.adobe.com/products/xd.html

Sketch

https://www.sketch.com/

Canva’

https://www.canva.com/

HTML & CSS

HTML5

https://developer.mozilla.org/en-US/docs/Glossary/HTML5

CSS

https://developer.mozilla.org/en-US/docs/Web/CSS

Flexbox

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box
_Layout/Basic_Concepts_of_Flexbox

CSS Grid

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout

Media Queries

https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

Sass

https://sass-lang.com/

CSS Frameworks

Bootstrap

https://getbootstrap.com/

Tailwind

https://tailwindcss.com/

Materialize

https://materializecss.com/

Bulma

https://bulma.io/

Foundation

https://get.foundation/

JavaScript

JS Docs

https://developer.mozilla.org/en-US/docs/Web/JavaScript

The DOM

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Async

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

Fetch API

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Extra Tools

Git

https://git-scm.com/

Markdown

https://www.markdownguide.org/

NPM

https://www.npmjs.com/

Yarn

https://yarnpkg.com/

Chrome Devtools

https://developer.chrome.com/docs/devtools/

Firefox Devtools

https://firefox-dev.tools/

Front-end Hosting

Netlify

https://www.netlify.com/

Vercel

https://vercel.com/

InMotion Hosting

https://www.inmotionhosting.com/

Namecheap

https://www.namecheap.com/

Front-end Frameworks

React

https://reactjs.org/

Vue

https://vuejs.org/

Angular

https://angular.io/

Svelte

https://svelte.dev/

TypeScript

https://www.typescriptlang.org/

UI Kits

Styled Components

https://www.styled-components.com/

MUI

https://mui.com/

Chakra UI

https://chakra-ui.com/

Vuetify

https://vuetifyjs.com/en/

Angular Material

https://material.angular.io/

Svelte Material UI

https://sveltematerialui.com/

Testing

Jest

https://jestjs.io/

Cypress

https://www.cypress.io/

Mocha

https://mochajs.org/

PyTest

https://pytest.org/

PHP Unit

https://phpunit.de/

MS Test

https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-wit
h-mstest

Selenium

https://www.selenium.dev/

Server-Side Rendering

Next.js

https://nextjs.org/

Remix

https://remix.run/

Nuxt.js

https://nuxtjs.org/

SvelteKit

https://kit.svelte.dev/

Angular Universal

https://angular.io/guide/universal

Static-Site Generators

Gatsby

https://www.gatsbyjs.com/

Gridsome

https://gridsome.org/

11ty

https://www.11ty.dev/

Jekyll

https://jekyllrb.com/

Hugo

https://gohugo.io/

Headless CMS

Strapi

https://strapi.io/

Sanity

https://www.sanity.io/

Contentful

https://www.contentful.com/

Ghost CMS

https://ghost.org/

Graph CMS

https://graphcms.com/

Wordpress

https://wordpress.org/

Animation & Charts

Motion UI

https://zurb.com/playground/motion-ui

Framer Motion

https://www.framer.com/motion/

Animie.js

https://animejs.com/

Three.js

https://threejs.org/

Chart.js

https://www.chartjs.org/

D3

https://d3js.org/

Greensock

https://greensock.com/

No Code Tools

Webflow

https://webflow.com/

Wix

https://www.wix.com/

Shopify

https://www.shopify.com/

Programming Languages

Node.js

https://nodejs.org/

Deno

https://deno.land/

Python

https://www.python.org/

PHP

https://php.net

C#

https://docs.microsoft.com/en-us/dotnet/csharp/

GoLang

https://go.dev/

Ruby

https://www.ruby-lang.org/en/

Java

https://www.java.com/

Kotlin

https://kotlinlang.org/

Rust

https://www.rust-lang.org/

Scala

https://www.scala-lang.org/

R

https://www.r-project.org/

Swift

https://www.swift.com/

Server-Side Frameworks

Express

https://expressjs.com/

Fastify

https://www.fastify.io/

Koa

https://koajs.com/

Nest.js

https://nestjs.com/

Django

https://www.djangoproject.com/

Flask

https://flask.palletsprojects.com/

Fast API

https://fastapi.tiangolo.com/

Asp.net

https://www.asp.net/

Blazor

https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

Gin

https://gin-gonic.com/

Ruby On Rails

https://rubyonrails.org/

Sinatra

http://sinatrarb.com/

Spring

https://spring.io/

Play

https://www.playframework.com/

Rocket

https://rocket.rs/

Databases

Postgres

https://www.postgresql.org/

MySQL

https://www.mysql.com/

MS SQL Server

https://www.microsoft.com/en-us/sql-server/sql-server-2019

SQLite

https://www.sqlite.org/

MongoDB

https://www.mongodb.com/

Firebase

https://firebase.google.com/

Supabase

https://supabase.com/

Redis

https://redis.io/

GraphQL

https://graphql.org/

Apollo

https://www.apollographql.com/

Authentication

JSON Web Tokens

https://jwt.io/

Oauth

https://oauth.net/

Passport

https://www.passportjs.org/

Bcrypt

https://www.npmjs.com/package/bcrypt

Full-stack Deployment

AWS

https://aws.amazon.com/

Heroku

https://www.heroku.com/

Digital Ocean

https://www.digitalocean.com/

Linodehttps://www.linode.com/

Apache

https://www.apache.org/

Nginx

https://www.nginx.com/

Docker

https://www.docker.com/

Kubernetes

https://kubernetes.io/

Vagrant

https://www.vagrantup.com/

Image & Video

Cloudinary

https://cloudinary.com/

Amazon S3

https://aws.amazon.com/s3/

Mobile App Development

React Native

https://reactnative.dev/

Flutter

https://flutter.dev/

Ionic

https://ionicframework.com/

Xamarin

https://dotnet.microsoft.com/en-us/apps/xamarin

Desktop App Development

Electron

https://www.electronjs.org/

NW.js

https://nwjs.io/

Python Tkinter

https://docs.python.org/3/library/tkinter.html

Web3

Web3 Foundation

https://web3.foundation/

Blockchain

https://www.blockchain.com/

Ethereum

https://ethereum.org/en/

Solidity

https://docs.soliditylang.org/

Algos & Data Structures

Codewars

https://www.codewars.com/

Leetcode

https://leetcode.com/

Project Euler

https://projecteuler.net/

CoderByte

https://coderbyte.com/

Web Assembly

https://webassembly.org/

70+ Job Find Websites For Developers

https://medium.com/@traversymedia/70-job-find-websites-for-devel
opers-other-tech-professionals-34cdb45518be

Management & Lifestyle

Notion

https://www.notion.so/

Trello

https://trello.com/en-US

Project.co

https://www.project.co

Zoho Projects

https://www.zoho.com/projects/

Agile

https://www.agilealliance.org/agile101

Scrum

https://www.scrum.org/

Waterfall

https://www.projectmanager.com/guides/waterfall-methodology