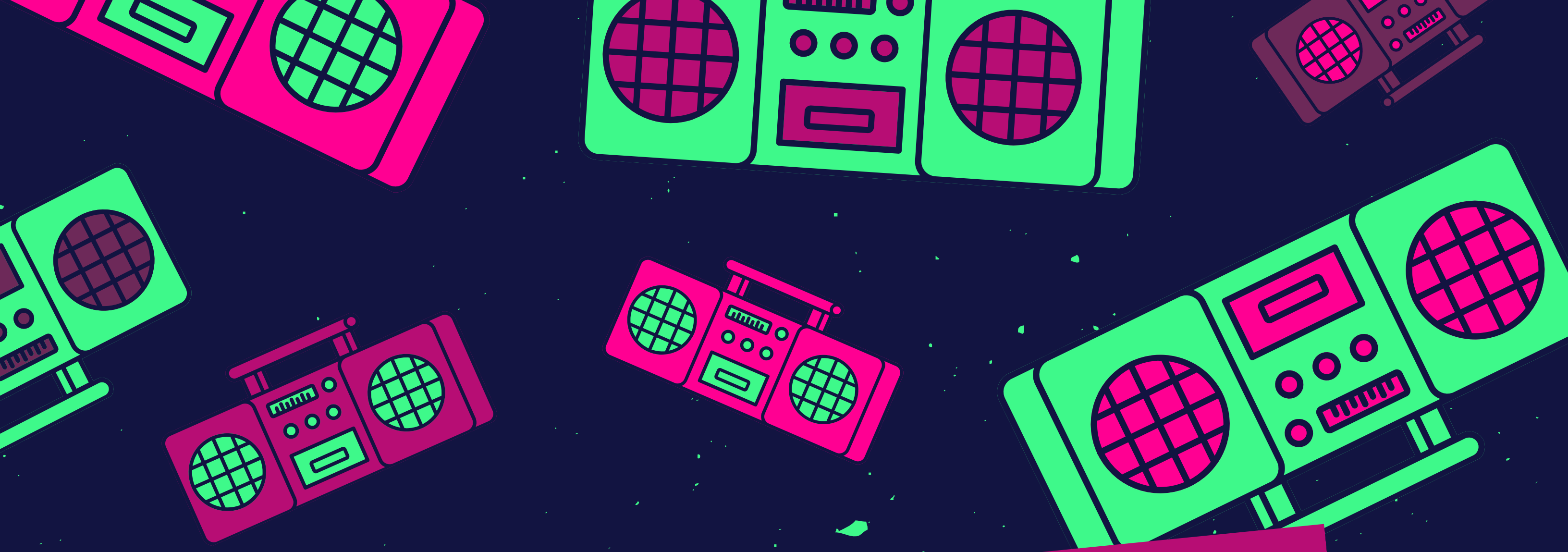


5 PROJECTS TO BECOME A Qualified WEB DEVELOPER

DEVON CAMPBELL

5 Projects to Become a Qualified Web Dev

Become Job-Ready by Doing Work

by Devon Campbell

In this ebook, I'll take you through five projects that will make you qualified to work as a web developer. Once you complete them, you'll be able to build awesome web-based software as a pro.

This book is not a step-by-step guide to completing each of the projects. (That would be a much longer book.) Instead, it's intended to be a starter for each of the projects. I'll give you the ultimate goal of the project, resources you'll need, and a brief overview of the very first steps. I expect you to pick up the torch and run with it from there.

In that way, it mirrors the actual career. Being a web developer is about understanding the end goal and mapping the steps to get to that goal. You won't practice just coding. This is also a great chance to practice figuring out those paths to completion.

Another part of being a web developer is asking the right questions. If you come up with any as you're working through these projects, please email me at devon@raddevon.-com. I'm always happy to hear from readers and help out when I'm needed.

If you like the book, please tell a friend about it by pointing them to WHATEVER THE URL ENDS UP BEING so they can get their own copy.

Project 1: Copy Your Way to the Top

Good artists copy, great artists steal.

This quote is often attributed to Pablo Picasso. I wonder if he came up with it himself. 🤔

The same principle applies to web developers. *This* project is about getting the **"good"** part down as a web developer by copying. You'll become **great** later when you steal what you learned and apply it to your own original work.

Copy a Simple Site

Take a basic site you're familiar with. Google and Craigslist are great places to start. Make your own lookalike from scratch.

Take a Deep Breath

I'm not asking you to build the search engine that changed the internet. I'm not asking you to build the world's largest classified ads site. I'm asking you to build a site that *looks* like one or both of those but doesn't necessarily *behave* like them.

What You Need to Know

- Web basics
- HTML
- CSS
- How to use a text editor

This one is pretty basic. You don't need an encyclopedic knowledge of HTML or CSS to get started. That's one point of this exercise: to shine a spotlight on the gaps in your knowledge and force you to start filling them.

Resources

I love the gentle intro in Josiah Spence's *HTML & CSS Guidebook*. Start at *What is a Website?* to give you some much-needed context that most resources skip. Then, go through the sections on HTML and CSS.

As you're working through the project, you'll want to have some good references at hand. Get familiar with the MDN Web Docs. ("MDN" stands for "Mozilla Developer Network." That's the same "Mozilla" behind the Firefox browser.) It's the best reference for HTML, CSS, and even Javascript. (You can skip the Javascript section for this project.) Their HTML element reference and CSS reference will help get you through the project.

Practice Your Tools and Processes

As you hone the skills you're here to hone (HTML and CSS), this is a great opportunity to practice using your tools. You could build on CodePen or CodeSandbox, but you'll learn more if you use the tools you'd use to do real work.

- Use your text editor. If you don't have one and don't know what to use, Visual Studio Code is a good place to start. (I wrote about why I recommend it.)
- Practice with a version control system. This probably means Git unless you have a specific reason to use something else. Git is tough to get started with, so it's important to practice as much as you can. (I recorded a tutorial on how to use Git and talked about how to write good commit messages.)

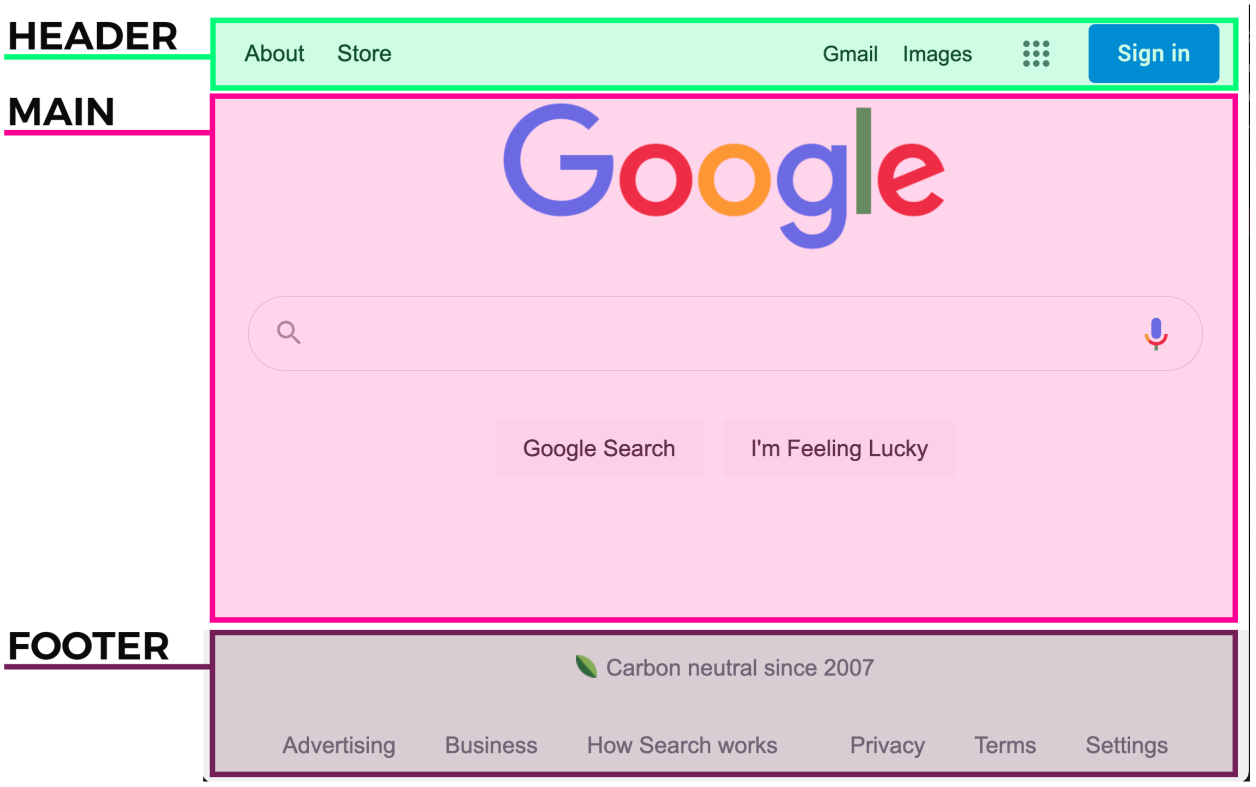
- Deploy that bad boy! This is a step people skip. It seems hard and scary if you've never done it, but you should deploy every project to get extra practice.

All that said, if you're completely unfamiliar with all of these steps, pick one or two per project. If you can't figure out any of them, do the project any way you can. Any practice is better than no practice at all.

Start With the Markup

You may be tempted to fire up your text editor, add a single element to your page, and style it until it looks right. Instead, I'd encourage you to focus first on getting the **semantics** of your HTML markup correct.

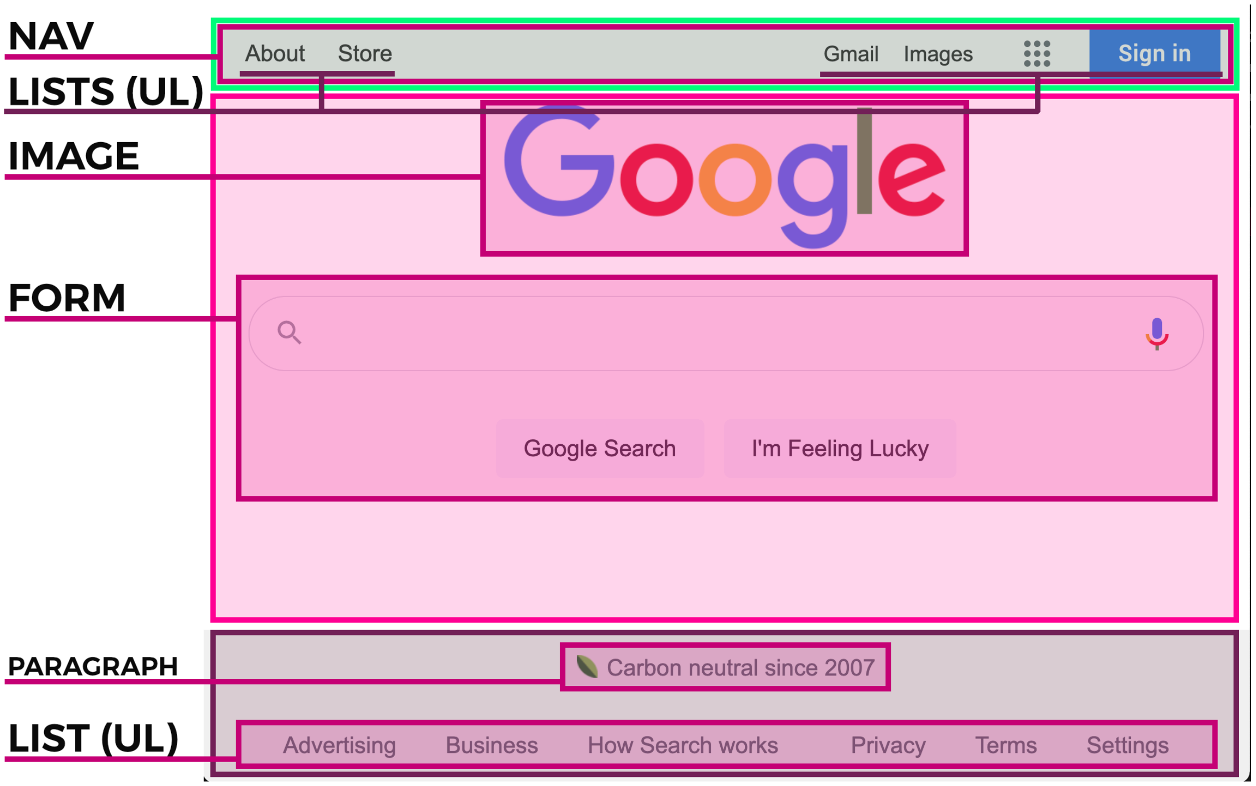
Here's what I mean by that. Your HTML is not about how your page looks. It's about the structure of your content. I start at a high level, defining the content's landmarks. If I were building the Google home page, that breakdown might look like this:



Google's home page broken into header, main, and footer landmarks

Keep in mind that this is *my* interpretation of the content's structure. That doesn't mean it's the *right* one. Look at the page you're replicating and decide what you think the content represents. Code up these landmark elements first. Then drill down further, breaking those landmarks into their child elements. Keep going until you're at the bottom of the document's hierarchy. For example, the header might contain a nav. That might contain two different lists (`UL` elements). Each of those contains several items (`LI` elements). Those each contain either a link (`A` element) or a button (`BUTTON` element).

Here's one more level of my breakdown:



Google's home page broken further into a nav and two lists contained in the header, an image and form in the main area, and a paragraph and list in the footer.

Now, Style!

You're ready to write your CSS now. Choose your CSS selectors carefully. They should be specific enough to target only the elements you want. but don't add more specificity than you need. I recorded a screencast to help you choose good selectors.

Need More Guidance?

If you more help, I recorded a video about how to approach cloning Google that will get you started on the right foot.

You can repeat this project building other sites. Once you're more comfortable with HTML and CSS, it's time to move on. Since these are foundational technologies, you'll practice HTML and CSS in every project. You'll get plenty of practice. Your next job-ready project will take you into Javascript land!

Project 2: Make Something Dynamic

You can know the combination for a lock and still fail to open it... if you don't enter the combination in the right order.

Learning web development is the same way. Learners get excited about front-end frameworks and try to learn them first. Since the frameworks sit on top of a language they don't know, they have set themselves up for a rough time.

I'm not trying to be a gatekeeper. I don't claim you must have mastered Javascript before you touch a front-end framework. At the same time, knowing Javascript before you try to build using an abstraction of it is useful. That's exactly what front-end frameworks are – an abstraction on top of Javascript.

Build a Habit Tracker with Vanilla Javascript

Build a basic web application using only vanilla Javascript. Pick a project that will let you practice a range of Javascript concepts. Make sure your project is small and tightly scoped. Check out my basic build of the habit tracker to understand the features of the app and how it works.

Quick clarification on one of the terms I'm using here: when I say "vanilla" Javascript, I'm using "vanilla" to mean something like "plain." That's Javascript without a front-end UI framework or library like React or Vue. Using lighter libraries for specific purposes (like persistence of data, for example) is fine.

If you search for "vanilla Javascript," you'll find a site for a framework called "Vanilla Javascript." This is a joke some programmer put together to try to prove a point that you didn't always need to use a framework. If you download a build, it's nothing – no code whatsoever. It exists only as a way for grognards to get a chuckle while it confuses newcomers.

No Longer a Copycat

The last project was to copy a web site, but this project is *not* to copy my app. Replicate the functionality in a way that makes sense to you. Build your own design. That could mean trying to replicate mine or creating your own from scratch.

What You Need to Know

- Web basics
- HTML
- CSS
- How to use a text editor
- Front-end Javascript basics

This project will exercise your HTML and CSS while also allowing you to write Javascript. You don't have to know Javascript inside and out. Here are some Javascript concepts you should know:

- the available data types, when to use them, and how they work
- how to manipulate data – doing math on numbers, manipulating strings, mapping arrays, etcetera
- how to interact with the DOM
- how to loop, manage control flow, and handle errors
- how to use browser APIs in your app
- how asynchronous code works

Resources

The Modern JavaScript Tutorial is a great free beginner resource for learning the language. You'll ultimately want to go through the whole thing, but you can get started with these bits:

- the "JavaScript Fundamentals" section
- the following lesson on debugging in Chrome would be very useful since you're not likely to get everything right on the first try
- "Error Handling" so you can tell your users when something goes wrong
- the section on the DOM which will let you make live changes to the page
- "Introduction to Events" so you can do things when users interact with the page
- "Forms, controls" so you can provide a way for the user to give you data
- You might be able to get by without going through the "Data Types" section, but it covers some helpful concepts

Depending on how much you want to build out the app, you may want to look at these bits as well. (I'll detail some of the optional components of the app further down.)

- LocalStorage to persist habit data in the browser
- The section on asynchronous code
- The section on third-party libraries

I present The Modern JavaScript Tutorial as my default choice because it's good and it's free. It's not the *only* option though, and learning for free isn't always the best way. Check out these alternatives:

- I love Wes Bos's Beginner JavaScript course. If you learn better watching videos, this is an excellent option. Wes has a great personality, and it's a lot of fun learning with him.

- A friend of mine Zell Liew has put together one of the most thoughtful and comprehensive Javascript courses I've ever seen. He also offers a free roadmap if you just need someone to point you in the right direction.

If you want to store the data in a database, work through Google's Firestore quick start tutorial from Google. If you want to support more than one user, do the Firebase Authentication quick start. (See the "Set Your Difficulty Level" section for more on these optional objectives.)

Even after you've learned, you'll still need references. MDN is a one-stop shop for a a comprehensive Javascript reference. For a quick reference, try my cheat sheet. It has practical examples of basic Javascript concepts. Refer to it when you forget how to use a feature of the language.

Here's how you'll get started.

Lay Down Your Markup

Since the application is dynamic, start by deciding which elements will always be present. Those will go into your HTML. You will render everything else with Javascript.

Put those foundational elements into your HTML. Give yourself a way to hook into them with your Javascript so that you can add the other elements you need. (Adding an ID to those elements is a safe bet.)

Set Your Difficulty Level

This project could be easy to build (and practically useless), or it could be more difficult and more like a real app.

You may want to build to the first difficulty level for your first iteration. Then, you can go back and layer more functionality to hit one of the higher difficulty levels. If you're confident, skip to the hardest difficulty and build that from the start!

I'll be using the Wolfenstein 3D difficulty levels to describe the difficulties of this exercise. I happen to have four levels, which matches up perfectly with Wolfenstein.

Can I play, Daddy?

Habits are not persistent. The user enters them and they show up on the page. When the user refreshes, they are gone.

At this difficulty, your primary concerns are:

- rendering the proper elements onto the page
- handling the events the user will fire to change data on the page
- making those changes to the data

It's a great exercise, but it's not very practical. Not many users would get value out of a habit tracker that doesn't save the habits and the tracking anywhere.

That doesn't mean it's not valuable to build. You wouldn't include it in a portfolio, but it could be appropriate practice for your skill level.

Don't hurt me

Add persistence using LocalStorage. This is the easiest way to persist data. Be aware that the data will be persistent only in the same browser on the same machine. If a user switches from Firefox to Chrome or to another computer, their data will stay where they created it.

This difficulty level could be useful. It's not what people are accustomed to these days. Most people expect their data in a web application to follow them around. This build would work like a local application, assuming the user stays in the same browser.

Bring 'em on!

Swap your persistence from LocalStorage to the Firebase Firestore database. Firebase is a range of simple cloud services offered by Google. It includes a database called Firestore. Using their web SDK, front-end developers can add database to their apps without a ton of hassle.

The free tiers are currently generous enough that you could build this app and never pay for any service. If you go public with it and it gets popular, that's another story.

For this version, store your habits in Firestore so that you can see the same data from any computer. The app is getting closer and closer to a "real" practical app at this point. The only thing left is to add user authentication so that different users can use the app. At this point, everyone's habits go in the same bucket, and there's not way to log in. The next difficulty level handles that.

I am Death incarnate!

Keep storing your data in Firestore, but add authentication using Firebase Authentication. This lets you support multiple users. Firebase Authentication does for authentication what Firestore does for databases. It's a quick and easy way to add user accounts without being a back-end developer.

Once your habit tracker supports user accounts, you'll need to change the way you store habits. Each user's habits will go into a bucket that's only accessible by that user. The app needs to pull the correct data for the current user.

You have a full-fledged app at this point. You could release it, and people who need a habit tracker could use it. Congratulations on making it this far!

Think about your UI

It's time to write some CSS! This time, focus on designing a great UI. Steve Schoger put together a great series of UI design tips into a Twitter moment (whatever that is 😆). Read through these, and apply them to your design of this app. You'll improve the design by at least 10x.

Stuck?

If you absolutely can't finish the project and have exhausted all your resources, you could skip ahead and take a look at the finished "I am Death incarnate!" build. Try to avoid it, but don't beat yourself up over it if you have no other way forward.

If you do skip ahead and look at the finished project, try to come up with a new project and apply what you learned on it. This will help you lock in your learning.

Now that you've built an app the hard way, it's time to learn why people are excited about front-end frameworks. They make building web user interfaces much easier!

Project 3: Build with a Front-End Framework

> *Tools are for people who have nothing better to do than think things through and make sensible plans.*
>
> *— Laini Taylor, Muse of Nightmares*

Front-end frameworks are tools for building user interfaces. Now that you've experienced the tedium of building a dynamic app with only Javascript, you can fully appreciate the advantages of a front-end framework.

The question of which front-end framework to use is not simple to answer. It depends on your long-term goals for learning web development. If you have an industry or region you want to work in, you should find what the prevailing framework is in that area.

If you're open to starting your career as a freelancer (Here are some reasons you should consider it.), you've got a lot more freedom. Consider Vue because it's easy to learn, has a huge ecosystem of libraries, and has tons of people using it. Having lots of users means it will be easy to find help. I recorded a video on building your first app in Vue (version 2) if you want to get started quickly.

If Vue doesn't excite you, you make the call. I have a video to get you started with React too, but there are tons of other options. **Whatever you do, don't let this decision**

paralyze you. The front-end frameworks share a lot of common ground. If you decide later you've learned the "wrong" one, it will be even easier to learn the "right" one.

Port Your Habit Tracker

The nice thing about porting an existing app is that the only thing you have to figure out is the framework. You don't have to learn to use the framework while also designing and building a new app. The app is already done.

This will also make it obvious where the framework makes things easier and where it makes it harder. Port over the hardest version of the app you built for your last project. If you need a little extra challenge, step up to the next difficulty level and layer that on top.

After you've ported your habit tracker, build new apps in your framework of choice to continue to level up. I wrote an article on how to generate your own project ideas that could help with this.

What You Need to Know

- Web basics
- HTML
- CSS
- How to use a text editor
- Front-end Javascript basics
- Framework of your choice (or Vue if you can't decide)

Resources

The most popular front-end frameworks have pretty good documentation. You can use their "getting started" tutorials to get up and running with the framework quickly. They usually have a tutorial that gives you the basics too. Here are handy links to the official tutorials for the three hot ones:

- Vue Essentials
- Intro to React
- Angular Tutorial

A quick note before we continue: if you're in camp Angular, be aware that Angular went through a major paradigm shift a while back. Popularity of Angular dropped off significantly after this cataclysm. If you're searching for help on Angular and find a resource you want to use, make sure it's covering the version you're using. Old Angular is commonly called "AngularJS" or sometimes "Angular 1," but if the resource was written when AngularJS was the only Angular, it might refer to it as just "Angular." Yes, it's a nightmare, and I'm sorry you have to deal with this while you're learning. The resources referenced in this ebook will cover post-cataclysm Angular.

This is true to a lesser extent with other frameworks. If you're using Vue or React and try to use a tutorial or resource written for an older version, it may or may not work. It's not as bad with these because they haven't made any shifts of the magnitude of Angular 1 to Angular 2. Just be aware of this when you're searching and when you're asking for help. The version you're using is very important.

If you want to take the next step with the frameworks and go beyond the official tutorial, here are some resources I recommend:

Vue

- *Vue - The Complete Guide (w/ Router, Vuex, Composition API)* is almost universally acclaimed. I don't have personal experience with any of the Vue courses unfortunately. Through research, I found this video course by Maximilian Schwarzmüller is far and away the most recommended course. People praise the author's infectious enthusiasm that keeps you engaged with the material. It's very comprehensive and has just been updated to cover Vue 3.
- *Learn Vue for Beginners* is a great free option if you're ready to go beyond the official tutorial. It's on YouTube, so you're going to suffer through some ads. It's a small price to pay for what you get here though. My favorite thing about this tutorial is that it also covers deploying the app. That's a critical step that most free tutorials skip.

React

- **Dave Ceddia's free** *Learn the basics of React in 5 days* **email course** is a great place to start after the official tutorial. Most React courses teach you React… and React Router… and Redux… and a bunch of other stuff. Sure, you probably want to know those eventually, but it's so much easier to learn this one piece at a time. Have you heard this old adage: "How do you eat an elephant? Shove the whole damn thing in your mouth at once!" No, you haven't, because that doesn't work. The real answer is "One bite at a time." With this course, Dave gives you the first bite of React. If you like

Dave's style and want to go deeper, he has a book called *Pure React* that will get you there.

- **Wes Bos's *React for Beginners*** is a great follow-up to Dave for people who learn well with video. This one isn't free, but it's worth the cost of admission. You'll build a fun project, start to finish, with Wes. He takes a "batteries included" approach, teaching you to use commonly used libraries. You're ready for that if you already did Dave's course.
- **The Odin Project has a free course on React** that curates other resources from around the web and ties them together.

Angular

- Just like Vue, another Maximilian Schwarzmüller course – ***Angular - The Complete Guide*** – came up over and over whenever people discuss recommendations for Angular courses. I haven't tried it, but people rave about the quality of both the content and the instruction.
- Angular University gets a lot of love, and their ***Angular For Beginners*** course is available for free. You can also sign up with them to get access to quite a few courses and ebooks.
- I bought ***ng-book*** way back when I learned AngularJS. It was an easy read that gave me a gentle introduction to my very first front-end UI framework. If books are more your speed, this new version updated for the latest version of Angular is a great choice.

Again, you don't have to choose one of these frameworks. You need to consider your goals and which framework will get you there faster.

Once you're ready to build, here's how you'll get started.

Map Your Habit Tracker Onto the Framework

After you understand best practices around how your framework structures apps, plan how the habit tracker maps onto that. For example, Vue (like many other frameworks) is all about reusable components. I'd start by thinking about where the UI should be broken up into separate Vue components.

Once you've done this, go ahead and start migrating your app to the framework.

Look for Framework-Specific Libraries

If you're using libraries in your habit tracker, look for versions of those libraries wrapped for use in your framework.

For example, I used the Firebase official Javascript SDK in my habit tracker. A quick search turns up Vuefire, a Firebase SDK wrapper for Vue that allows me to create reactive data bindings between the UI and the database. In other words, when data changes in the database, the UI is updated to reflect that. (As of this writing, it's only available for Vue 2, but that's only temporary.)

This wouldn't be necessary for some libraries, but for libraries that would force you to write code to connect the library to your UI, check to see if someone has already done that work for you.

Deploy!

This is a step so many new developers skip. Building a web app that runs only on your machine means **you're not finished!** Finish the job and put it up somewhere. Knowing how to do this is a critical career skill you can't afford to skip on every project.

The most straightforward way to deploy this site is to deploy to Netlify.

Stuck?

Find communities around the framework you've chosen. These communities come in many different forms.

- Slack/Discord communities
- IRC channels
- Forums/subreddits
- In-person meetup groups

You're also welcome to reach out to me. I'm not an expert in any front-end framework, but I have enough experience I can help with a wide variety of issues. I also have friends who know more than I do that I can ask for help!

That's another good point: it's great to have relationships with people you can fire off a quick question to when you don't know where to turn. Don't try to go make friends *after* you have a problem. Having a first interaction with someone where you're asking them for something is a bad look. Build relationships early and think about how you can add value for that person. There are ways even though you may not have the same amount of experience they do. This makes it easy for them to reciprocate when you're in a jam.

StackOverflow is another option as a last resort. In my experience, this community is openly hostile to newbies. You have to ask the question in exactly the right way to get an answer. You'll be marked as a duplicate in no time if you're remotely close to an existing question. When you're asking a question here (or anywhere really), you want to:

- show that you've put in some effort
- clearly describe all the circumstances around the problem
- isolate the specific issue as much as possible
- make it super easy for people to answer

Remember that you're not paying these people and they don't owe you an answer. With those facts in your head, it will help you orient toward gratitude and making things easy on them. Watch my video for more help on writing great questions about your code.

Next up, you're venturing into the deep dark waters of back-end development.

Project 4: Write a Back-End

You can build interfaces now, but interfaces to what? If you went deep on the previous projects, you've been able to build full-featured apps without writing back-end code. That's a cool superpower, but it will only get you so far. Eventually, you're going to want to build an app that does more than just store data. For this, you're going to need to write back-end code.

Build a Back End for Your Habit Tracker

Now it's time to build a back end for your habit tracker. If you completed the Firebase difficulty level in the previous projects, you should have a pretty good understanding what your app's back end needs to do. It needs to:

- store data in a database
- return it to your app
- (potentially) allow for users to authenticate with your app

You can just rip out Firebase and Firebase Auth and build those functionalities in your own code.

If you haven't taken the app that far up to this point, your app – *and* your skills – are about to grow up fast.

You're going to write a service for your existing app to consume. You'll get practice writing server-side code, interfacing with a database, designing an API, and consuming an API.

What You Need to Know

I'm going to break this down a little bit more here because in each area you have several options. The resources I curate will focus on my recommended option. If you don't have a strong preference, go with my recommendation. Otherwise, feel free to go in any direction you choose.

How to Write a Web Server

You could do this in any one of many different languages. If you're only experienced with Javascript, use NodeJS . It lets you write Javascript to run server-side. I'd recommend using ExpressJS. It's a framework that makes it easy to write a web server in Node. Express is mature and has an enormous community you can tap for support when you need it.

If you are more experienced with another language like Ruby, Python, Java, or C#, feel free to write in those.

REST

REST is a style of API that gives you some good parameters to work within while writing your web server. (API === web service.) The API is what you'll use from your front end to either *store* stuff (like add a new habit or increment the counter on an existing one) or *get* stuff (like a list of a user's habits).

Databases

You will store your data in a database. For the purposes of your habit tracker, the data consists of the habits and counts as well as the users of your app. Having this in a database will make it easy to store and retrieve.

You can use a relational database or a NoSQL database. The habit tracker is simple enough that either one would work for you. If you have no preference, use MongoDB. It's a NoSQL database that is ubiquitous and easy to use.

Authentication

Authentication is what allows users to log in to your app. You may have already implemented authentication on one of the previous projects... *but* you've only implemented it on the front end. Google provided you with an API to connect you with all the back-end code you needed. Now, it's time to write that yourself.

If you took my suggestion to use ExpressJS for your web server, look at PassportJS for authentication. You don't need to use Passport for authentication – you could just write your own – but authentication is an easy thing to get wrong. Passport has already gotten it right, so don't try to re-invent the wheel.

Resources

Since there are so many different ways you could build this back end, I'm not going to focus the resources on one of them: Node for your web server, Mongo for the database, and Passport for authentication.

Node

Download and install the latest LTS (long-term support) release of Node to get started.

From there, the *Introduction to Node* guide on the official site will help you understand what Node is and how it works.

MongoDB

Install MongoDB. Once that's done:

- read up on how to structure your data
- set up authentication on your local instance
- learn to connect to your local MongoDB instance

Now that you're able to connect, you'll want to learn how to perform the basic data operations.

- insert
- read (including queries and compound queries)
- update
- delete

Passport

Read the overview section of the official Passport docs to get it installed. Then you'll want to go through at least the Authenticate, Configure, and the Username & Password sections of the docs to get a handle on how it works and how to implement it.

You could use other means of authentications with Passport like Facebook or Twitter, but username and password logins are the easiest place to start.

Putting Them Together

Learning about each of the tools you'll be using to build the app is great because, if you try to learn them by putting them together in step one, it can be harder to understand where one stops and the other begins. Lines get blurred.

On the other hand, learning Node, Express, Mongo, and Passport individually does not really teach you how to build an app that uses all of them. For that, we have a couple of great courses that take you through building an entire app!

Set up an Express.Js App With Passport.Js and Mongodb for Password Authentication is a solid tutorial to build a simple app putting all the pieces together. It's text-only, but it is free and does a great job as an introduction to building an app with this tech stack.

Learn Node is not free, but it's a video course and it's more comprehensive. It's also taught by one of my favorites: Wes Bos. Like his other courses, it's fun, engaging, and packed with great information. This course goes well beyond what you'll need to know to build out this project.

Now that you have the foundational knowledge you need, here's how to move forward.

Start From Your Previous Project

Work from the version of your habit tracker built with a front-end framework. You'll start by ripping out Firebase. Then, you need to write your own services to replace it. Use the two resources in the previous section to help understand how you can do that.

Think of this version of your app as two projects that work together. You'll build an API that handles authentication, storing data, and retrieving data. You should be able to make HTTP requests to that API without any sort of front end.

Then, you have your front end which makes requests to your API.

Read Up on HTTP Response Codes

Your API needs to respond with the correct response code so that it's easier to deal with the response in your front end. Check MDN for descriptions of all the response codes and familiarize yourself with what's available to you.

This isn't *strictly* required, but please be a good internet citizen and use meaningful status codes. It will make your life easier too.

Testing an API

You could write you entire API before start to finish before then trying to plug your front end into it, but that would be crazy. Instead, you'll want to test your API as you're writing it.

You don't want to write the whole thing and then try to layer another thing on top of it that will also need to be tested. If you have a problem, is it the API? Is it the front end? You'll have to find out before you can even start debugging.

To test your API, you can use an HTTP client. Here are a few you can try:

- HTTPie- This one is command line based. If you're on a Unix-based OS, you already have cURL, but it's not all that easy to use. HTTPie is.
- Hoppscotch- If you prefer a GUI client, this is a nice choice. The cool feature here is that you can easily save your calls into collections. You could create a separate "collection" (the app's name for folders of saved requests) for each API you want to test and save requests for each endpoint and method.

- Paw- If you're on a Mac and don't mind paying for software, this is my favorite API client. It has the saving features of Hoppscotch and lots more. It does a great job handling authentication and will export your requests to code in a language of your choice. It's a native app, so performance is great too. You can buy it on its own or get access to it with Setapp.

Deploy!

This will be your toughest deployment challenge yet, but **do not skip it!** There are 1,000 different ways you could deploy this, but here's a quick recommendation for beginners:

- Back end- Heroku
- Front-end- CloudFlare Pages (You could just use Netlify like you did on the last project, but why not try something new?)
  - Vue
  - React
  - If you used a different framework, you can still deploy to CloudFlare Pages. Just check out the Vue or React tutorials. Your project is already built, so you'll just need to know the command you run to build. Set that command properly, and you should be good!

Stuck?

The same advice applies: find communities around the tech you're using. Look for:

- Slack/Discord communities
- IRC channels
- Forums/subreddits
- In-person meetup groups

My email is always open to you, so let me know if you get stuck and can't find answers.

This is where the rubber meets the road. In your next project, you're going pro.

Project 5: Do It For Real

At this point, you've done a little bit of everything. You just need to do more of it. Keep your skills sharp by building personal projects, but you don't have to *just* build personal projects. You can also start getting work.

Get Paid to Build for the Web

Here's what you may be tempted to do:

- Build a resume
- Add all those hot technologies you used in a hot "Skills" section
- Start applying for every Amazon opening you can find

You can do this if you really want to, but you're probably going to be laughed off the remote interview. You just don't have enough experience yet. They want 5 *years*, not 5 *projects*. Even if you drop down a few tiers and start applying for engineering roles at a smaller company, you're going to have a hard time finding one that wants to take a chance on you at this stage.

The problem is, everybody wants 3-5 years of *professional* experience. They *don't* want 3-5 years noodling around on personal projects in bedroom. So, how do you get that 3-5 years of experience if no one wants to hire you without it? Most people keep applying for jobs for a few months, assume the door has closed – there's no longer any way into the industry – and give up.

I went through the same thing when I got started, and I solved it by picking up paid projects. This allowed me to get several years of professional experience and then get hired into a great senior position. Now, it's time for *you* to do that too.

The really cool thing about doing this is that, unlike other methods of getting into the industry, you get paid while you're doing this one. That gives you a few advantages:

1. Instead of watching your savings run out while you wait for someone to hire you, you're actually making money. You've basically gone from a short runway to one that's as long as you need it.
2. It doesn't even have to be a runway. You may find you like the flexibility and control you have and that you want this to be your career. It can be.
3. If you do decide you want to go to work for someone, you have all the bargaining power. You don't have to tolerate any terms you don't like when you're negotiating with a prospective employer. If they don't want to give you the terms you want, you can just keep making money the way you have been.

If you think you're still not ready, just remember this. Even though you're in the infancy of your career, and you know very little compared to a senior dev, you know *way* more than the people who need your help. Very few companies have even a single software engineer on staff. Almost all of those same companies need a web site or other software you could build for them.

Someone once told me that, to be an expert, you just have to be one lesson ahead. Keep this in mind, and sell that expertise you have by being one lesson ahead to the businesses that need it. As long as you're not trying to mislead people, you can gain experience while you make money making businesses run better.

What You Need to Know

I realize these are going to sound intimidating, but they're really not as bad as you think. I'm going to give you all the resources you need to make it happen.

- How to start a business
- How to get clients
- How to get paid
- How to save for taxes

Resources

I wrote a a free email course on setting up your freelance business. That will help you start the business and figure out the financials. Unfortunately, this is only available for my US readers.

If you're outside the US, find a local accountant. Find one who works with small businesses or freelancers and can tell you what you need to do to start your business and how to handle taxes.

Getting clients is another subject entirely. There are lots of ways to do it. I had my own strategy: picking a niche and forming relationships within that niche. It worked out pretty well for me. Here are some videos that will help you do it too:

- Become a Web Developer Roadmap- Step 5: Networking (Relationships are make or break!)
- Start Getting Paid as a Web Developer
- Get Web Dev Clients While on Lockdown

Getting paid is easy if you have good clients. I wrote an article on how to get better clients that will help make sure you aren't working with people likely to stiff you. Once you've got the right clients, it's just a matter of making it easy for them to pay you. I do that using Harvest.

Oh, the Things You'll Learn!

By doing work in the real world for real businesses, you're learning so much that it's hard to learn any other way. Being a developer is way more than just writing code. If that's all you've done, you're really missing out on a lot of skills you need to do this work well.

- You're going to have to think about *why* you're doing what you're doing. (Not why you personally are doing this, but why there's even work to be done. Why does a company hire a web developer? It goes well beyond, "Duh, to build a web site!")
- At some point, you're going to be in a position where you're working with another developer. Maybe you're just talking to a departing developer to understand an existing system you'll be working on. Maybe you're actually collaborating with an in-house developer in real time.
- Most of my work happens on legacy code, not brand new greenfield projects. If I didn't learn (on the job, no less) to read legacy code, I'd be borderline worthless. By doing work in the real world, you'll get to have this experience.
- You'll probably get to contribute to an open source project, and not in some contrived forced scenario where you set out just to do that. You'll be contributing because you found a great library that does 99.9% of what you need it to do for the current project, but that last 0.1% is a dealbreak-

er. Is it easier to rewrite this library yourself from scratch or to add the feature you need and make a pull request? The answer is easy for me, and it soon will be for you too.

This Doesn't Have to End Your Job Hunt

I get it. Your heart is set on working on the big engineering team. You don't need to give that up. Scale back your job application efforts to half what you're currently doing. Devote the other half of that time to doing paid work now.

New developers applying for jobs at FAANG (Facebook, Apple, Amazon, Netflix, Google) are like new basketball players learning how to dribble, pass, and shoot and then showing up to Lakers tryouts. You're just not ready, so don't waste all your time trying to make it happen.

It's no harm to keep applying while you're doing your paid projects, though. In fact, it's helpful because you won't know when you *are* ready – only the companies you're applying to will.

Do More Than One

One paid project is not enough! The intent here isn't that, after a single paid project, you're now ready to stop and devote all your time to applying for jobs. Keep your momentum going. Keep building your experience.

Stuck?

You need to find other freelancers and ask them for help. I happen to be another freelancer, so feel free to ask me.

You're a Web Developer!

I promised only to make you qualified to be a web developer, but you actually became a real-life pro in the process. Your journey isn't over, but, by following this method to get started, you've flipped the script.

Most people learn everything they need to learn and then beg companies to let them be web developers. Instead, you've taken matters into your own hands and made *yourself* a developer.

Where you go from here is your call (and that kind of power over your circumstances is unusual, I've found). You're making money, so you can keep doing that. If you're itching to be in that open office, you can keep applying and ask for exactly the pay and benefits you want. (If they decide to play hardball, you can go right back to doing your paid projects on your terms.)

Whatever path you decide to take, I hope it's a gratifying one. Email me if you get stuck along the way or even just to let me know how it's going. 🤘