Coder Manual

Job Hunting Handbook

Coder Manual

Job Hunting Handbook

Author: Rob Dey

Website: http://codermanual.com

Table of Contents

Preface

At this point, you've hopefully completed a full Coder Manual course and have real, practical web development experience. If so, congratulations! You've accomplished quite a bit and you should be proud. You're ready to step into the world of professional web development and you're in for a treat. You'll hopefully find that being a web developer can be a great experience - fulfilling work, high salaries, great work/life balance, flexible companies, half-day fridays, free beer (if you're of age) are not uncommon on job listings, if you know where to look.

I've written this ebook to help you apply what you've learned and earn money. I strongly recommend that you read through it all the way through - it is full of tips and information that will round out your understanding of being a web developer and enable you to hang with the seasoned developers more comfortably. Don't worry, this ebook is designed to be an easy, quick read.

I. Have a Strategy

Now that you have development experience, you may be interested in a number of ways to apply your new skills. Perhaps you want to work full-time, part-time, free-lance, or start your own business.

This chapter will "demystify" your options and help you make good choices as you move forward.

An important note: everyone's skill level will vary based on experience and ability to absorb new concepts. The best way to get better is to write more code. That said, a large part of a web developer's job is to tackle new things, i.e. solve problems they've never faced. You'll hear many people say that half of the job of a web developer is using Google to find solutions. Even if you have experience building multiple applications at this point, you need to embrace the fact that, you'll often face problems you've never coded solutions for - the real skill of a programmer is to be able to make the computer work for you no matter what task is at hand. The point I'm making is that, while you should always be conscious of your current skill level, be fearless in your approach to new things and understand that your job will often be to solve problems you've never encountered before. Don't let it demotivate you early on if you get stuck on problems - everyone goes through that at some point, and you'll get better and better with each new problem and solution.

Also note that your learning doesn't necessarily end as a web developer, because of the rapid evolution of technology. Try to be comfortable with the idea that you'll be learning new things fairly regularly. The benefit is that your value, and thus salary potential, increases as you learn.

Here are a few suggested strategies for moving forward:

1. Seeking Full-Time/Part-time Work

There are two common approaches I want to point out.

a. Join a company/startup and work on a single company's code.

b. Join a digital agency and work on a variety of projects for multiple companies. Digital agencies are basically outsourced teams that do web/programming/marketing work for multiple companies at the same time. Hashrocket (http://hashrocket.com) is a well known shop that might fall under this category.

c. Find and connect with recruiters and let them do the work of getting you hired (e.g. http://www.cybercoders.com).

There are pros and cons for each approach. Let's start with **(a), joining a startup or established company**. If you go with (a), I recommend that you first choose the company based on an alignment of interest and values (we'll discuss how to find these companies in a bit). What I mean is that you should work for a company that you believe in - every company generally has a mission. Don't just join a company for the money, or who pays the most. Even if you're strapped for cash, do your best and hold out for the company that is closest aligned with your own beliefs. The good thing is that there has been an explosion of new companies, not just in Silicon Valley (or New York, or any of the large cities where tech is flourishing), but all across the globe. People have caught wind of the startup movement and the momentum is really on the move - which

means you have plenty of options (if you know where to look). Tech has grown in virtually every industry, so whether you're interested in healthcare, helping the poor, getting rich in the final sector, or anything in between, there's a decent chance that there is a job waiting for you. Take a hard look at your own interests and try to find a company with the best match.

The reason why this is so important is that, once the excitement of the new paycheck dies down, your work/life balance and happiness at work will be based on the work you're doing for the company. Make it worth it for yourself - you'll become a better coder, and have meaning in your day-to-day work/life balance. You'll be happier.

You'll also likely develop a sense of camaraderie with your teammates so try to get to know the team before you join a company. Try to find a team that you can learn from. Interview them when they interview you. Get a sense for what they've accomplished, who else they've worked for, and how sophisticated their current approaches are to web development. Every team handles things differently, and some are more sophisticated than others, sheerly based on experience.

In contrast, going with **(b), joining a digital agency**, will at least give you exposure to many different people and companies (who will be your clients at a digital agency), and therefore you'll get exposure to many different ways to building and maintaining web technology.

In terms of money, even as an entry level Rails developer (no previous work experience), you should be able to find a company or startup that pays $60-$80k for full-time work. Some companies pay even more than this for entry level programmers. According to the Bureau of Labor Statistics, the median pay for software developers (programmers) in 2012 was $93,350. If you do a Google search for things like 'rails developer entry level salary' or 'rails developer average salary' you can sift through many sites to get a sense of the market value of your skill set. Keep in mind, every resource gathers their data differently, so variance will be apparent, but the BLS metrics are usually a good benchmark.

The main reason you should go with a digital agency is to build experience. If you have the mindset of putting aside your passions and values to build experience (which is totally respectable), then going with a digital agency is a good choice.

Digital agencies are usually small or medium sized companies that are hired ('outsourced') by small businesses to Fortune 500 companies for their technology and/or marketing needs. You would likely join their IT staff to handle web development. They sometimes have a team of designers who work closely with you to create the web experience that the client is looking for. Digital agencies also usually have a salesperson or sales team that goes out and gets work from clients. You'll have to get good at 'scoping out the work' to estimate how many hours each project will require before you get started, so that the sales team can propose costs to the clients.

Digital agencies are abundant across the globe and are often hiring actively or are passively on the lookout for new talent. Since many digital agencies work on projects for Fortune 500 companies, this is a good way to get first hand experience with that level of establishment. You'll build a wealth of skills at an agency since every project will probably be very different from the last, and you'll also get a sense for efficient work flows.

The downside here is that you're usually not working on a project that you're passionate about on a personal level. On a professional level, you may really enjoy your work, but how meaningful it is will vary.

Finally, you could go with **(c), connecting with a recruiter** for them to assist in placing you at a company. Sometimes if you fill out a LinkedIn profile (you can set up a free account at http://linkedin.com) completely and state that you have Rails experience recruiters may contact you directly. Once you build more experience, you'll find recruiters contacting you regularly. You can also reach out to recruiting firms - search Google for things like 'Rails recruiters' or recruiters in the nearest big city, you should be able to find companies. From there just be honest - if you're inexperienced let them know you want entry level positions, but still be descriptive about what you've built (i.e. e-commerce experience, HTML, CSS, and so on).

2. Become a Freelancer

You may already have a great job in a specialty outside of web development or you may just want to get your feet wet before you dive in and start committing larger portions of your time to web development. If that is the case, freelancing may be perfect for you.

Working as a freelancer can be fulfilling because you're still your own boss and you have a lot more control over when you want to work and who you want to produce a product for. It makes for a flexible lifestyle.

The downside is that it is usually up to you to seek out the work, and if this approach becomes your livelihood, you need to make sure you're able to find work regularly (we'll cover this in a bit).

The great thing about freelancing is that in 2015 and beyond, almost everyone needs a website. Whether it is a local shop or a friend, lots of people are in need of help with getting a website running and most people don't know how to do that. If you know where to look, you can keep yourself very busy.

Being a freelancer requires a lot of organization. Not too much, but just enough to get things done right. You'll have to wear many hats in your role. Freelancers will often team up with other web developers or designers to split up the work. If you choose this route, you should really value good, clear communication. This is one of the most important parts of the job and it really helps you in all aspects of career growth as a free-

lancer. You'll also need to be conservative in your estimates of time and scope of work (we'll discuss this later), and only take on work that you feel confident about producing well.

3. Start Your Own Business

If you think you've got a great business idea, you might want to consider building a business. We can't get into the full breadth of subtopics related to starting a web business here, but I'll point you in the right direction. People have many different goals and expected outcomes when starting businesses, but since the first dot com bubble and the latest wave of startup culture, there are a few sort of 'universal truths' - if you will - that have surfaced. Personally, I've built my own profitable business (http://DJCoursesOnline.com), but not without experiencing failure first (more on failure in a bit).

Here are some things to think about before starting a business:

a. Be Driven by Passion (as cliché as it sounds, it's true)

Starting a business is challenging, but if you're truly passionate about your company mission, you may increase your odds of success. Your company needs to align with your personal views. You should take a hard look at yourself and make sure you're dead-honestly passionate about what you're doing. The reason you need this is because you will inevitably work long hours and this passion is literally like the fuel in your car - it is what will keep you going. Remember that most businesses fail. Passion will drive people to do crazy things and it takes 'crazy' sometimes to achieve great results.

b. Solve a Need

Your business may never get off the ground and become profitable if it doesn't solve a need. Again, you need to take a hard look and see if it really does. A good strategy is to test the potential before even starting the business - survey a number of people to get significant feedback and sentiment about your solution to a problem. Would people pay for it? Is it scalable? It is very easy to get a sense of momentum in your mind about an idea that you think is cool or has potential, and that makes it easy to convince yourself that your business will work. Again, most businesses fail, and you need to be comfortable with that possibility. The more your business solves a real need that people experience, the less there's a likelihood of failure.

c. Growth Doesn't Happen on its Own

Some people launch websites thinking that traffic will just come right after launching. A physical store in a strip mall in a small town has better odds of getting traffic than a website. Marketing your site is a very real and necessary task in order to see growth. You may eventually get ranked in a search engine's results list but not until you have traffic in the first place. Search engine algorithms change regularly to become smarter at detecting which sites people care about visiting (i.e. sites

with good content) vs. sites that don't add value to people's day or lives. So the first step after launching a business is to get people through the door. You should consider press as one of your top priorities. You could pay for ads or try to market your site all by yourself, but, press is one of the best, most cost-effective ways to go. If you're solving a need, the press will naturally want to cover your product.

d. Criticism, Failure, and Adaptation (Pivoting)

Society, sometimes places very strange views on lots of things, two of those being criticism and failure. Contrary to what most people will say, criticism and failure are very powerful agents in your path to success. They are *good, useful* things. You should always seek negative feedback and criticism because you simply can't look at your own product with the same perspective as others. They will show you things that you simply don't see. You don't always have to agree, but you're better off building a list of observations from others than being clueless.

Regarding failure, use every single experience of failure - big or small - as a lesson. Break it down as if you're in a classroom and really look at what caused the failure and how you can do better going forward. If you become a student of your own failures, you'll grow personally and build better businesses. Also, if there is a chance of failure, try to let it happen early and often. You'll learn more up front, and you can usually risk more earlier than later. You should get comfortable with the fact that many ideas can or will fail and that you should move past them quickly. It is very easy to get into the mindset of becoming attached to a business idea, and not wanting to let go for a year or longer. That is valuable time you could spend learning and testing new things. Most successful entrepreneurs you've heard about have had many failures before their most popular success stories. Finding a business model that works is like throwing things on a wall to see what sticks - throw lots of things and something will eventually stick. It may sound counterintuitive and fairly childish, but it is generally true.

II. Presentation Layer

Before you start reaching out to companies, you need to have your 'presentation layer' looking great. If you're seeking work as an employee I'm referring to four things here:

a. Résumé

b. Portfolio/Code repository

c. Cover letter

d. Web presence

Let's start with (a). Your résumé should be short - no more than one page unless you have 5+ years of relevant work experience that simply can't fit on one page. If you have no prior web development experience, focus on previous experiences

you've had that could make you a fun person to work with and show that you can exceed expectations. Remember that being a web developer is really like being a problem solver who uses code as the medium for the solution. If you can show that you will really dig to solve problems and not give up easy, you'll be telling employers that you have the right traits.

Regarding (b), if you've completed a Coder Manual course at this point, you should already have pieces of a portfolio and real experience building production grade aspects of applications. Having a live portfolio site that an employer can visit before or during an interview will help. Often times, employers want to see your Github account or whichever repository site you use. Therefore, you should take that seriously and keep pushing public projects or small applications that you build to your Github account - the site tracks your activity over time so you may want to avoid a cold streak before applying to companies just to stay fresh.

In terms of (c), you should always add a cover letter whenever possible if you're applying for a job. Even if there is no place to add one in the application, find an email address of a relevant person at the company to send it to. Try to keep it short and snappy - just a few paragraphs. That increases the odds that employers will read it all. The cover letter is your chance to convey a few key things that will get you in the front of the line before interviews even happen: you can show that you're a great communicator by writing well; you can use your real speaking voice in terms of diction for employers to get a sense of your personality; you can express interest in the company; and you can point to your best projects quickly before they've reviewed your résumé.

The format I like to follow:

- Opening line after your greeting should go something like: "I'm very interested in the Rails developer opportunity and I want to tell you why I may be a good fit."

- Then, the first paragraph should briefly list the relevant languages and technologies you have experience with, such as HTML, CSS, JS, Ruby, Rails, etc. You should also mention a quick line about your personality such as, "my friends know me as a passionate individual" or "my colleagues think of me as a very uplifting kind of guy/girl." Just be honest about your top personality trait.

- The main body paragraph should list one, maybe two, things about the employer's current web technology, mission, or other reason that makes you want to join them. It could be that they have really cool interfaces and you're passionate about contributing to them, or that they are a music startup and you make music, etc. Then, you should mention some of your projects and embed the links to those projects in the text itself; if you struggle with finding overlap between what you've built and how it would pertain to the company, try to be creative and dig in to your coding experiences or even your workflow and work ethic - you'll find overlap if you look

hard enough. The main point is to make sure you embed hyperlinks to your projects. This makes the cover letter a joy for the reader because they can click *inline links* to see your projects as your words describe the relevance. This creates a coherent experience for the employer. For example, you could say, “I’ve built and managed an application that serves DJ courses to paying subscribers on various subscription plans,” where part of that sentence links out to the project your describing. Most email applications allow you to embed hyperlinks in the text and this is an elegant way to show employers your projects. Even if you only have practice applications built, this will help you stand out as long as you explain your application’s capabilities descriptively.

- Finally, add a paragraph - just a line or two - that explains that you’re looking forward to a response, and that you’re interested in learning more about the team. Also provide the best way to be reached - ideally include your phone number to be direct. You want to make sure you state that you’re interested in ‘learning more,’ as opposed to flat out saying that you’re ‘the one for the job,’ in most cases. This shows that you’re experienced in that you want to make sure you like the team and the culture before you jump in - they’ll respect that.

In terms of (d), web presence, this refers to things like your LinkedIn profile and other social media sites. You may want to house your résumé on its own web page at yourname.com or yourname.org, etc. You may also want to put together a site that showcases your best work. A good looking portfolio will impress employers, and there are tons of free/low cost HTML résumé and portfolio templates, that are just a Google search away. Sites like http://about.me are also great solutions for creating personal web pages that look good. Again, a Google search will show you similar options for website builder applications. You should also have the skills to build your own fairly quickly since it should just be based on HTML, CSS, and maybe JS and/or Bootstrap. Just remember that your web presence represents the impression employers will have of you so keep it real and professional. If you’re building a **freelancing** career, a great looking website or portfolio helps. You can add to it over time as you get more clients. As mentioned above you should be able to find templates or build your own site.

III. Job Hunting and Outreach

1. Seeking Full-time/Part-time Jobs

As mentioned before, you should search for jobs that are aligned with your personal interests. If you’re talking to recruiters, let them know what your interests are. The excitement of working for a great paycheck will wear off quickly and you’ll be left with how much you really enjoy the work. The more meaningful it is to you the better. See if you can find companies with visions that match your own set of beliefs or simply meet your objectives. Your goal may be to build experience, or work with a specific kind of team.

Don't underestimate the number of jobs that are out there. What you see or hear about is likely just a sliver of what is out there - its easy to develop impressions about job markets based on friends and discussions, but it really is a massive market. Here are some of the best places to look for jobs:

- http://authenticjobs.com (great list of companies, allows you to filter for things like freelance, remote, full-time, etc.)

- https://weworkremotely.com (primarily lists remote jobs)

- http://stackoverflow.com/jobs

- http://www.simplyhired.com

- http://indeed.com

- http://flexjobs.com (another list of remote jobs)

- http://jobs.rubynow.com/ (mostly Ruby/Rails jobs)

- http://www.rorjobs.com/

Also, you'll find interesting job boards like https://nytm.org/made (which is an excellent site that lists hundreds of NYC-based startups, many of which are hiring and are open to remote work). There could be similar job boards for other big cities as well. Get creative with Google and search for companies directly. You never know what you may find. Never be afraid to reach out to a company directly and express interest, even if they don't actively state that they are hiring, because in 2015 and beyond, nearly everyone needs tech help. And regardless of which city their office is in, it's worth reaching out because of how many companies offer remote jobs for programmers.

Don't underestimate startups - they may be young companies, but funded startups can sometimes pay higher salaries than Fortune 500 companies while offering a modern or radically different company culture.

Read job listings carefully as employers often have key requirements of what to include in your application. Also, be sure to go through a number of job listings just to get a sense of what skills are required in general. You may see an emphasis on certain skills that you feel could be brushed up a bit more before you apply. That said, companies are often willing to train you if you meet most of the requirements and have a great personality.

Also, don't forget to ask around. Again, many people are currently in need of web help, so there is a good chance someone in your network knows of an open position.

2. Freelancers

You need to have a real strategy if you're planning on building up your freelance career. It can be as simple as looking through Yelp and calling local businesses directly, or hiring someone to help you. Obviously, someone who has great communication/sales skills is ideal for the sales role. Since this

isn't a sales guide, we won't cover those details here, but you shouldn't have too much trouble finding your first contract. Again, if you don't want to take the manual approach to sell to companies directly, you can always check sites like the aforementioned http://authenticjobs.com and filter for contract work only, or work with recruiters. Also, you should be able to find some contract work on Craigslist.

You'll also need to have a process nailed down. Once you land a client, the first thing you need to do is:

a. Have a conversation. Your goal should be to extract as much detail from the client about exactly what they are looking to have done. Keep in mind, there will likely be a loss in translation. What I mean is that most of your prospects and clients typically won't use technical jargon that you should be used to using at this point. Clients will also often under/over-estimate the amount of work involved. Understand this and keep things very clear as you proceed.

b. Provide the client with a formal 'Scope of Work' or 'Proposal.' Those are just formal names for a document that outlines the work and costs involved. You'll need to describe, briefly, what is involved in each step and how many hours it will take. For example, if you're building a contact form for the client, you'll want to break it down into specific sections and give each section a time estimate like "*0.5 hours - View file. We will create the markup for your contact form and link the form fields to a contact object using embedded Ruby.*"

Be conservative in your time estimates. You always want to exceed expectations rather than let them down, and you never know when you'll run into an unforeseen problem. I've heard the rule of thumb is to multiply the time you think it would take for a given task by 2x or 3x. That way the client is happier if you get the work done quicker as opposed to the other way around. You'll also need to clearly display your hourly rate and the total cost for the work (include any costs you may have to incur like hosting, API's, etc.). Generally freelance Rails developers in the U.S. charge between $55-$150/hr or more depending on experience. If you've got very little experience, you may want to start low and work your way up as you gain more client experience.

c. Communicate often and clearly. The client will appreciate this. And always we remember to remain courteous even if the client is an @$#%*&%! They're out there - you've been warned.

d. Get paid. Write up an invoice for the work when it is complete and send it over to the client. You can find high quality invoice templates from a quick Google search, that come in various formats like Microsoft Word or Google Docs if you don't have access to Word.

IV. Acing the Interview

First of all, break down whatever you've been told about interviews before. It isn't something to get nervous or too excited

about. It is simply a step in the path between you and a company. Unless the company states otherwise, you should think of it as a conversation, not something so formal (obviously maintain a professional appearance), and it is just as important that you interview the company as much as it seems all about you.

There are at least two types of interviews in the web world. One is the typical interview where you talk to the company and each find out if it's a good fit (can happen over phone, video chat, or in person). The other is a technical interview, which usually happens if the employers value a formal or theoretical computer science education - basically, they'll quiz you to see if you can solve some programming questions. It isn't always a deal breaker if you have trouble answering the questions, but your other qualities would really have to shine in that case.

The typical, conversational interview is by far the most common but every now and then you'll run into a technical interview. Here are a few pointers for the typical interview:

a. First, if you happen to have a mellow personality, try to amp it up a notch. It is hard for people to tell how interested we are unless we really take the enthusiasm up a notch. Obviously, be genuine in your enthusiasm, and if you followed the strategy tips mentioned previously, this shouldn't be a problem since you should be talking to a company you're excited to be a part of anyway.

b. Learn how to find and convey value in the experience you have, no matter how much or little you do have. When describing a previous project, you could phrase it like, "I built a Rails app that takes payments," or you could say it more like, "I have experience building a Rails application from front to back. The app supports e-commerce functionality that I integrated from scratch - my users can sign up for free or pay a monthly fee and automatically get billed via recurring charges. I also integrated the Sendgrid API to give the application the ability to send out email notifications and I've deployed the application to live Heroku servers. You're welcome to interact with it here: xyz.herokuapp.com." In other words, the same idea can be delivered in multiple ways, so be thorough, thoughtful and descriptive. When you have less experience, its important to make whatever you do have really shine. That said, be honest in your descriptions - you don't want to set their expectations higher than you can meet or exceed.

c. Interview them from the start. Make it clear from the beginning that you're interested in learning more about the company during the interview and that you have questions that need to be answered. In particular, ask them about the team. By default, you should be interested in knowing what the team is like, how experienced they are, how many people are involved, who you have to report to regularly, etc., but explicitly asking about these things will make them respect you a lot. Asking about the team is a good way to

show real interest because it shows you've thought about how things will work out on a personal level, beyond just the professional level. Also, ask about the company culture - put them on the spot and ask them what the day-to-day energy feels like or ask them to describe a typical work day.

d. Emphasize that you're a fast learner and deliver on that promise. You don't have to be brilliant to follow through on that, you just have to be persistent. Plan to take notes regularly on the job, so that your teammates don't have to repeat things - especially in your first year, you'll have to maintain passwords, learn new processes, etc., so the other developers will appreciate you being eager to stay on top of things.

To prepare for technical interviews, you can search Google for things like 'ruby on rails interview questions' to get a sense of what might be asked, but note that the sites you'll find will vary widely in terms of what they think are common technical interview questions. So don't let those questions scare you, they come in a wide variety of difficulty ranges. Remember, even if you had no prior experience, if you've successfully completed a Coder Manual course, you've got proven chops by now, since you've built real, practical applications and that should be the most important thing when you're getting a job - not how well you can remember a certain corner of the ruby language. That said, you can't always anticipate which companies will attempt to run a technical interview, but it doesn't hurt to brush up on your skills, especially when you're just starting out. Of course, with such high entry level salaries (for a Rails developer, according to Payscale.com, that is $77,322 at the time of this writing), why not just start by seeking an entry level position and working your way up? You'll be much more stress-free if you find a job where you're expectations are lower, which may be a good thing if you're just starting out.

Finally, an important note. On some level, you need to think about the chances of getting a job, how much it will pay, and how great the culture is, all as a numbers game. The companies you find and the interviews and outcomes you have will be a sliver of what is out there because of pure probability. You need to do your best to take the emotion out of this process. Don't get too excited or feel too let down with each win or loss. Just keep moving and trying until the right job finds you. Once you're able to view it as a numbers game and pure probability, you'll find that you'll reach the outcome you want more easily. The point is that, if you keep looking, and you shouldn't have to look too hard or too long with the skills you have, you'll find what you want because this job market is in your favor right now.

V. Further Learning

Being a web developer, your learning usually doesn't end because of how quickly technology changes. Embrace that fact. You'll see job listings evolve over time as well to match these changes. So here are some great resources to keep you sharp and relevant (note that some links may change or stop working over time as the web changes):

1. RailsCasts - http://railscasts.com - Lot's of free videos on highly focused Rails topics.

2. Rails Guides - http://guides.rubyonrails.org - Excellent written material on the ins and outs of Rails.

3. Ruby Koans - http://rubykoans.com - Great resource for keeping your Ruby chops sharp. I highly recommend this, and it may be tricky to set it up at first, but it is totally worth it.

4. The Ruby on Rails Tutorial by Michael Hartl - https://www.railstutorial.org - Absolutely amazing resource. You can read the book online for free and you build a Twitter clone along the way. This is a great way to learn Test Driven Development, which many companies want in candidates. If you're serious about learning Rails, do this tutorial!

5. Codecademy - http://www.codecademy.com - A fun, easy-to-use website where you can brush up on everything from HTML, CSS, Javascript, jQuery, Ruby, Rails and more. I strongly suggest that you complete their courses on all of these topics as you should be able to fly through them pretty quickly.

VI. General Tips

I wanted to include a few general tips to help fill out your understanding of web development:

a) Remember that, especially when you're starting out, you'll find yourself Googling for answers and suggested solutions to code problems you face. There is absolutely no shame in this and many of my colleagues joke that half of the job is Googling. A common workflow is that, whenever you run into an error, you can copy and paste the actual error message itself in a Google search. If it's not an error, just Google the key terms that describe the problem you're facing like 'rails controller create params.' You'll find that http://stackoverflow.com will show up in the search results quite often. This site will be your new best friend so get comfortable using it often.

b) The best way to keep your skills sharp and see career growth is to build more things. I can't reiterate this enough. I find that the motivation to keep building things comes from working on projects that you find to be useful, practical, or are passionate about. This goes back to what I mentioned before about finding jobs that are aligned with your personal interests. That said, you don't always need to work for money. Consider building projects for fun - things that you think would be fun and useful for yourself. For example, you could build a mini Rails application that connects to the YouTube API and gathers new videos about your heroes like Elon Musk or Larry Page (or whoever your heroes are). You could refer to the API tutorials at http://codecademy.com if you don't know where to begin. Also, consider contributing to projects, especially open source projects. You can find

great open source projects, especially those backed with great moral or ethical goals.

c) Something I like to do is keep an eye on job listings over time. Checking them regularly will give you a sense of what the market is like and what skills are in demand. You'll also get a sense of how other companies set up their IT architecture so that you can make improvements on your own turf.

d) Keep your code clean and well written. Add comments wherever relevant and be thorough. Also consider using tools like http://www.jslint.com or http://validator.w3.org to get instant feedback on your syntax quality.

e) Start exploring software design patterns. There are plenty of books about them that are specific to each language. These will increase your awareness on how experienced developers handle common programming problems from a theoretical standpoint that can filter down to the practical level easily.

f) Some cool links:

- CSS Tricks - http://css-tricks.com - Well written/managed resource for handling common CSS problems and learning new tricks to really impress your friends!

- Chrome Experiments - http://www.chromeexperiments.com - A site that shows some pretty amazing work from developers around the world.

- Dribbble - https://dribbble.com - Show and tell for designers.

- Awwwards - http://www.awwwards.com - Lists high quality web sites.

VII. Glossary

I've included a 'glossary' section to list out key terms that you may come across as you get your career off the ground. These are just definitions seen through my own lenses, but I've done my best to keep them objective.

- **app**: Short for application and can refer to any software application though it is often used to describe applications designed for mobile devices like iPhones or Android phones.

- **bug**: An error or problem in a computer program that causes it to produce an unexpected result.

- **CSS**: Cascading Style Sheets. Used for adding color, positioning, layout control, font-size, and more to markup like HTML.

- **database**: An organized set of data. Common systems for managing databases include MySQL, PostgreSQL, SAP, Oracle, NoSQL, Redis, MongoDB, CouchDB, etc.

- **David Heinemeier Hansson (DHH)**: The software developer who created Rails. Involved with 37 Signals, a popular software company.

- **design patterns**: Refers to reusable solutions to common programming problems.

- **Donald Knuth**: An influential computer scientist, author, mathematician and professor. His work, *The Art of Computer Programming* is well known/revered by computer scientists.

- **HTML**: Hypertext Markup Language. Used for putting content on a web page like text, images, or embedded videos.

- **Javascript**: Not to be confused with the Java programming language, Javascript is a programming language that allows one to control interactive properties of web pages. Javascript has recently risen in popularity even though it has been around since the days of Netscape, especially because it can be used as a server-side language as well. Frameworks like AngularJS, Backbone, Meteor, and tons more are based on Javascript.

- **jQuery**: A popular library that makes writing Javascript easier.

- **responsive**: A term used to describe websites or web applications that are optimized to look good across browsers or devices of any size/width. CSS media queries are one way to control how things look across various devices.

- **Rails**: A framework, or a way of organizing code, which makes building web applications with Ruby easier.

- **Ruby**: A programming language that is commonly used for building web applications.

- **server**: A computer that is optimized to serve web page content via HTTP responses (or another protocol) for browsers or other applications. Companies like GoDaddy, Heroku, Rackspace, Amazon, and many more rent out servers.

- **Tim Berners-Lee**: The computer scientist credited for inventing the World Wide Web.

- **web application**: Usually a term used to describe a website that does more than just display content and actually have functionality built in that possibly interacts with a database. Examples are Facebook, Hulu, Dropbox, etc.

- **web server**: Software that resides on a server that handles the actual delivery of the HTTP response (or other protocol). Common web server libraries include Apache, Webrick, and Unicorn.

- **website**: Usually this term refers to sites that act more like ‘brochure’ sites in that they simply just display HTML or some other form of content without deeper functionality occurring in the background. The term is broadly used to describe any web property including web applications, so it can be seen as an umbrella term, as well.

- **Yukihiro Matsumoto (Matz)**: Chief designer of the Ruby programming language.

- Fin -

http://codermanual.com