

A Non-Technical Founder's Product Development Guide: How to Build Your First MVP

Mikey Squires, Princeton COS BSE '25

An efficient, process oriented, all-encompassing approach for bringing your product idea to the market.

Table of Contents

1. Introduction.....	3
1.1 Purpose and Scope.....	3
2. MVP Overview.....	4
2.1 What is an MVP, and why is it useful for entrepreneurs?.....	4
2.2 What does this mean for this guide?.....	4
3. Vertical Marketplace Web Application.....	4
3.1 What is a vertical service marketplace app, and why should I build it?.....	5
3.2 What features are necessary for my VSM Web App MVP?.....	6
4. Necessary Features.....	6
4.1 S-Tier - Essential.....	6
4.2 A-Tier.....	8
4.3 B-Tier.....	9
5. Example Project.....	10
5. 1 What is College Athlete Connect?.....	10
5. 2 What will the product service look like?.....	10
6. External Resources.....	11
6.1 Youtube - Specialization.....	12
6.2 Stack Overflow - Specific Coding Issues.....	12
6.3 LLM's - Coding Tutor.....	12
7. Technical Set Up and Planning.....	15
7.1 IDE's vs Text Editor.....	15
7.2 Google Workspace.....	16
7.3 Wireframing.....	17
7.4 Branding Tools.....	18
7.5 Software and Accounts.....	19
7.6 Browsers.....	20
8. Coding: Where to Start.....	20
8.1 HTML and CSS.....	20
8.2 Git and Version Control.....	23
8.3 JavaScript.....	24
8.4 Scalable Coding Practices.....	29
9. Front End Development.....	31
9.1 Organization.....	31
9.2 Recommended Development Ordering.....	32
9.3 Implementation.....	32
10. Back End Development.....	45
10.1 BaaS Platforms.....	46
10.2 Node.js.....	47

10.3 Setting up Firebase.....	48
10.4 Incorporating Firebase Tools.....	50
11. Custom Hosting.....	60
11.1 Acquiring a Domain.....	60
11.2 DNS Routing.....	61
11.3 Custom Mail Records.....	61
12. Conclusion.....	62
12.1 Market Testing.....	62
12.2 Final Message.....	62

1. Introduction

1.1 Purpose and Scope

In 2024, computer science education is more accessible than at any time in history. Between online guides, youtube tutorials, and AI tools, anyone with internet access and a desire to learn can quickly become sufficient in coding. Furthermore, SaaS products currently dominate the startup atmosphere, as the low cost, low risk model for development creates a low barrier to entry for aspiring entrepreneurs. There are countless free development tools available to the general public that can be used to launch a business. If you have a great idea for a product or service, and you want to build and monetize it, you can do it.

However, given CS is a vast field with so many disciplines of study, for someone with little to no coding experience, it's hard to know where to start. Furthermore, even if you have intermediate coding skills (like myself before writing this guide), and you are looking to develop your product, it's hard to know which languages, frameworks and tools are the best for you to add to your repertoire and utilize. Here we come to the core problem when learning CS, *options*. There are so many options on what to build, how to build, and what to use, it's extremely hard to decide what you *should* do and use.

When I first attempted to develop my own startup, I was disgruntled to find there was no complete start-to-finish guide on how to build a web app. I was eager to learn, but didn't know *what* to learn. In other words, you don't know what you don't know. That is the core purpose of this guide. A typical minimum-viable-product (MVP) build from scratch would involve significant time and research at every step of the process. Instead, I went through the trials and tribulations for you, developing my own MVP, testing/researching all possible avenues, to tell you what's best, and what mistakes to ignore.

The intention of this guide is to expedite the development of your MVP by telling you what (I believe) you should learn and should do. The needs of the market are ever changing and evolving, so speed is crucial. The faster you can develop an MVP, the quicker you can get market feedback, allowing your business to quickly adapt and pivot with its (scalable and sustainably built, thanks to this guide) infrastructure. Once your concept finds product-market fit (PMF), the tools from this guide ensure your infrastructure will be able to scale with demand. As you read through this guide, I will explain why these coding practices are crucial to the MVP methodology.

This is by no means an all encompassing guide, but it's more connected and leveled to the problems and struggles you will be facing when starting. I know how hard the process of starting is, because I was just there. My hope is to build the success of fellow entrepreneurs with proper CS education, and allow anyone with an idea the potential to build a business out of it.

2. MVP Overview

2.1 What is an MVP, and why is it useful for entrepreneurs?

“A minimum viable product, or MVP, is a product with enough features to attract early-adopter customers and validate a product idea early in the product development cycle. In industries such as software, the MVP can help the product team receive user feedback as quickly as possible to iterate and improve the product. Because the agile methodology is built on validating and iterating products based on user input, the MVP plays a central role in agile development” (<https://www.productplan.com/glossary/minimum-viable-product/>). The key words here are *validation* and *agility*. Your MVP is not meant to build the entirety of your business idea, because you don’t even know if your idea is good yet. It would be detrimental to spend months developing a product to find no one wants it. Give it the bare necessities, launch as quickly as possible, and validate your idea. When we build, we build for agility. Cutting shortcuts through the use of bad development practices may get your product to the market quicker, but is detrimental to your scalability and ability to pivot. You build with a backup plan in mind. I will explain throughout this guide how our coding practices relate to this methodology.

For a more comprehensive look at MVP methodology, I recommend reading Eric Ries’ book, *The Lean Startup*, which sets the baseline of many modern entrepreneurial and business frameworks.

2.2 What does this mean for this guide?

Simply put, this guide is meant to show you how to develop the bare necessities as quickly as possible. We will offer additional paths you could explore or spend more time on towards your MVP, depending on the time you are willing to sacrifice for an improvement in quality. But we won’t go down every avenue. Rather, we will walk through the processes we see as strictly necessary. This guide should serve as the backbone of your development practice, allowing you to branch off as you see fit.

3. Vertical Marketplace Web Application

For the starting point of our entrepreneurial journey, we believe the best starting point is a vertical marketplace web application. We believe this is conducive to new developers, broad enough to be applicable to a large user base, flexible enough to allow you to specialize, but still narrow enough that we can offer specific recommendations. We cover what each of these terms mean below.

3.1 What is a vertical service marketplace app, and why should I build it?

Let's break this term down into its individual components.

Web App - A web app is any software application that operates through a browser, such as Safari, Chrome, or Firefox. The useful part about this is that your application will be available on all devices that can connect to the internet through a browser, phones, tablets and computers. Unlike desktop or mobile applications, which run on the devices operating system, web apps will be more accessible and cross compatible. However, a web app will require internet connection.

If your service needs to run without internet connection, is very dependent on performance, or reliant on specific components of a devices architecture/operating system, you may want to consider a desktop or mobile app instead. However, these tend to be much more involved, and longer to learn with a higher barrier of entry for new developers. For simplicity and scope, a web application will be easier to develop and serve our purpose.

Marketplace - A digital marketplace is any type of platform that allows users to upload and sell a product or service, and allows other users to view or purchase those products/services. This may also be referred to as ECommerce. A lot of new startup ideas will fall into this category.

Furthermore, the market for Ecommerce is growing rapidly, with an expected CAGR of 9.47% in the next 5 years, and revenue of approximately \$5000bn by 2029 (<https://www.statista.com/outlook/emo/ecommerce/worldwide>). This sees a ton of potential for new startup founders.

Many new startups are focusing on social media or AI, but once again, these are going to be a lot harder to learn to develop. Marketplaces are a good mix of a business model that is still relevant, growing rapidly, with a relatively low barrier to entry.

Vertical vs. Horizontal - “A vertical marketplace is an online platform that specializes in a specific industry, sector, or type of product or service. Unlike horizontal marketplaces that offer a broad range of goods and services, vertical marketplaces focus on a particular niche, catering to specific needs and audiences.”

Examples of a vertical marketplace would be [Airbnb](#), specializing in property rental, or [Care.com](#), specializing in caregiving services. On the contrary, a horizontal marketplace would be something like [Amazon](#), which sells anything and everything.

Horizontal marketplaces are much more competitive and harder to break into, especially against the giants. Vertical marketplaces allow you to find a specific niche that may not be filled yet.

Furthermore, by specializing in a vertical niche, we significantly narrow the scope of our development. We know exactly what and who we are developing for. This sets a clear baseline for the features we need to learn to develop, which we'll discuss more soon.

For more specifics on the difference, and what your business idea may fall under, I recommend this: <https://www.sharetribe.com/marketplace-glossary/vertical-marketplace/>

Note: In our example project, College Athlete Connect (CAC), we are building a *peer-to-peer service* marketplace. We are connecting consumers offering a service to other consumers. Therefore, when building, we will need to support the needs of both types of consumers. More on this soon.

3.2 What features are necessary for my VSM Web App MVP?

For this guide, we are exploring the process of building a dynamic (not static) web application. Essentially, this means our website is going to be interactive, allow users to input data, and view others data. The look of certain webpages will change based on what data is entered. For a more comprehensive overview on the difference between static and dynamic websites, I recommend this: <https://www.wix.com/blog/static-vs-dynamic-website>.

Below we outline the features necessary to support your business model, ranked as S-Tier (strictly necessary, and covered extensively in this guide), A-Tier (next steps as you approach PMF, we may include some suggestions on how to integrate them throughout the guide), and B-Tier (Once you have found PMF, how to scale your product, not covered in this guide).

4. Necessary Features

Now that we know we have narrowed our scope to developing a vertical peer-to-peer marketplace, we can derive what features we need to incorporate for the purpose of the platform. We outline the necessary features for our MVP into 3 levels. The S-Tier sets the baseline of what knowledge we need and what this guide is built around. The A-Tier includes many important features that may help increase your product functionality as you seek to specialize and branch off from this guide. The B-Tier features should only come after this, and leaves you with a jumping off point for next steps to consider once you're finished with this guide.

4.1 S-Tier - Essential

When developing your MVP, we see the following features to develop as crucial for market testing. When it comes to vertical marketplaces, the following are essential to the core functionality of your product. Without them, it will be impossible to conduct accurate market tests to see how people interact with it, and what they think of your idea.

- 1. Appealing UI/UX** - By no means will this be the final, polished version of your product. In fact, it would be irresponsible to spend a lot of time on making it look perfect, just to

find out people don't want to use your product. However, you need to make sure it looks good enough, such that if people choose not to use your product, it's not a result of how it looks. It takes little effort to do things like adding appropriate branding colors, button effects, and other details that make your website feel much more official to a new customer.

2. **Profile Creation/Data Storage** - Given our product is a peer-to-peer marketplace, we need to add the ability to create profiles. If your MVP is meant to sell products, this profile could be the storefront homepage. Otherwise, as in our case with CAC, half the user base is marketing themselves as the product, so the profile will suffice. You will need to create fields for them to input their information, and then have a way to quickly store and fetch that information - meaning you will need some backend/server side solution.
3. **Login Functionality/User Authentication** - In our marketplace, half the users are buying/viewing products (or people), and the other half is selling. While guest profiles (no login) may suffice for the first half, the user base that is selling products will need a way to access and edit the information of their products. This involves using their email, or some other identifier, along with a password, and referencing that information against your stored information, to make sure only the owner of information can access their own profile.
4. **Security Protocols** - Even if you have proper user authentication, there are still smart hackers who can find ways to access information they shouldn't be allowed to by scanning or accessing your code directly. You can prevent this with simple, proper security measures in your back end database to make sure only authenticated users can access and edit data.
5. **Cloud Storage** - Chances are if you're just starting out, you don't own your own server. But that's fine, because there are companies that specialize in upkeeping their own large servers and allowing you to pay to hold some information in them. This allows you to store a large amount of user data, with it being immediately accessible and interactive towards the client side of your web app. This is where you'll store the user data and profile information we stored above.
6. **Public Domain/Hosting** - After you write all the code, you'll need a way for customers to actually access it. This can be done by acquiring a domain name, and using a service to host your website (html and other pages) and publish it to the internet. This will typically be the same as your cloud host for backend data.
7. **Traffic** - When using a cloud service for the backend, they will have limits on how much data can be written to, or pulled from, their servers within a certain time frame. Starting out when you have next to no users, this won't be an issue because the free tier of most cloud services should suffice. However, you want to select a service that will scale well with your business model, so you don't come to a point where you must transfer mass amounts of data to another, cheaper/better service.

- 8. Compatibility** - Whatever code you write, and tools you use, should be compatible with modern browsers. The digital world is constantly evolving, and you don't want your website to be suppressed or not functional because you didn't develop it using modern frameworks.
- 9. Organization** - Not so much a feature, but how you decide to organize your code is crucial. The number of different web pages you have will increase faster than expected, as well as images or other branding tools pasted into pages. You should have proper naming conventions and domains, so you can easily keep track of where your pages are, what they do, and to link them. It's easy to lose track of this for the sake of speed, but it will take twice as long to reorganize after the fact, as opposed to doing it correctly the first time.
- 10. Git Version Control** - Not a feature, but it's crucial that you are constantly saving your data and code to a repository. You won't be able to access old versions of your code directly within your code editor, it just saves the most recent version to your computer. If you make accidental changes that drastically change or impede the functionality of your website, you want to be able to revert back to old versions. In the same way that Google Docs allows you to access old versions of your prose, Git allows you to access old versions of your code.

4.2 A-Tier

Once you begin market testing, and start acquiring your initial user base, these are the next features you will need to work on developing. For some startup ideas, a few of these features may be considered essential depending on how users interact with the site. But for all startups, as the number of users increases, so will the amount of data you need to store and sort, which is why you'll need these.

- 1. Search/Filters** - Once your marketplace has a large amount of data for customers to sort through, whether its products, or in our case with CAC, athlete profiles, simply scanning through will not suffice and will deter new users. You need a method for them to search for specific products/people, and/or add filters based on specifications, controlling what the customer sees on the client side.
- 2. Reactive Design** - This is the process of making your design compatible with mobile devices, which have a different screen size and layout than computers. A useful tool for this is Flexbox, which can be used for mobile and desktop development, and could prevent you from needing to develop two separate versions of your site. Given the large amount of people that access the internet primarily on their phone, it's crucial you develop correctly from the beginning to make this transition easier.
- 3. Improved UI/UX** - This is up to the developer's discretion, but as you move closer towards your final product you want to extensively market, you'll want to make sure the design and feel of your website is catchy and optimized for an ideal user experience. This

may involve a complete redesign of your front-end, or changing a few elements based on user feedback.

4. **Increased Traffic Support** - As your user base grows, you'll need to be able to support more people visiting your website and uploading their data. If you follow the steps from the A-Tier about our recommended cloud solutions, this should come automatically without any extra effort (except having to pay beyond the free tier).
5. **CD/CI Workflow** - As you expand your team, and hire technical workers, you'll both be working on the same project. You need a way to both edit the same codebase without interfering with each other. Github offers workflow tools for this purpose, to see what versions/edits your teammates made, and what conflicts with your own. You can also test to see what your website will look like before you deploy it.

4.3 B-Tier

While the guide will not cover these features directly, we still want to leave you with a leaping off point as you continue your development journey. After you have a promising user base, and seemingly have acquired product market fit, these are possible next steps you'll want to consider as you continue to add features to your product:

- **Payment Integration** - We don't recommend attempting to monetize your service before it's been proven. If people don't want to use your service when it's free, they definitely (with a few exceptions) won't want to use it if they have to pay for it. But after you have proven yourself and gained some traction, the next reasonable step in scaling is acquiring revenue from a loyal customer base. You'll need to integrate with a payment service, and connect payments to your company's pay account. Carefully consider your payment structure - who is paying, what they're paying for, how much, and at what stage/point within the website you'll require them to pay. This step will likely involve multiple tests or pivots until you find the ideal payment structure.
- **SEO Tools/Analytics** - Oftentimes starting out, you don't want to promote your website. First impressions matter and leave a lasting effect, so it's better to limit your testing pool. But once you want to reach the largest audience possible, you need to utilize certain strategies to boost your pagerank in Google and other search engines. Analytics on how people access and interact with your website are crucial for this, which can be seamlessly integrated with Firebase, a BaaS we cover later in this guide.

5. Example Project

To aid in your development, it's useful to see how the methods in this guide can be applied to a real world product. For this reason, we introduce College Athlete Connect (CAC), which was developed through this guide's process.

5. 1 What is College Athlete Connect?

College Athlete Connect is a platform meant to enable high school athletes to find college athletes within their sport and proximity to coach them for private lessons. College athletes fill out a profile about themselves, with predefined fields about their skillset, location, coaching rates, and more. High schooler's or their parent's can then scan through and search for specific profiles that meet their coaching needs.

This falls under our definition of a peer-to-peer vertical marketplace, as users are selling a very specific "coaching service". It will also require all of the S-Tier features and functionality we mentioned, such as user authentication, profile creation, data storage, hosting, etc.

You can find the service here: College-Athlete-Connect.com

The repository of example code can be found here, which we include images of specific parts throughout the guide: <https://github.com/msquires22/IW-Final.git>

Note that it does not contain all sensitive information or files.

When considering your own business model, it is important to consider potential competitors. Competitors for my service include <https://www.coachup.com/> and <https://lessons.com/>. Given that this guide is feature oriented, it is important to consider the features of your competitors' service, and what features within your own product will be different or better to give you a competitive edge.

5. 2 What will the product service look like?

If you're unable to access the link above, here are some pictures of our service demonstrating the features we are implementing:

The screenshots illustrate the CAC platform's user interface:

- Top Left:** A grid of three athlete profiles: Tucker Poff (Crew, Cornell), Sebastian Garibaldi (Wrestling, Princeton University), and Mikey Squires (Wrestling, Princeton University).
- Top Right:** A detailed view of the Mikey Squires profile, showing his photo, name, sport (Wrestling), school (Princeton University), class year (2026), rate (\$30), email (msquires@princeton.edu), phone number (609-373-9330), and bio (Starter at 174lbs. Walk-on from Upstate NY.).
- Bottom Left:** A single-profile view for Mikey Squires, identical to the one above.
- Bottom Right:** An "Edit Profile" form for Mikey Squires, allowing users to update information like Name, Sport, School, Class Year, Rate, Bio, Email, Phone Number, and Photo.

This implements the mentioned features above, with user accounts and customizable profiles (our marketplace product offering) that you can scroll through and click on for more information.

6. External Resources

As previously mentioned, this is not an all extensive guide, but rather a complete process overview. We are not seeking to replace external resources, but organize them and link to them as we see fit, in the most efficient way possible. While writing this guide, I scanned through countless tutorials and videos to find the best, most succinct ones available, so you don't have to waste your time doing the same. However, as you're developing your product, you may find you want or need to use a language, framework or other technology not covered in this guide. Or perhaps you ran into a small coding block/problem and need to resolve it. The good news is that many other people have already run into the same problems you have, and there are likely a lot of resources to help you pass that block and acquire the necessary knowledge.

Nowadays, there are three free main resources available for resolving coding issues and learning new technologies: YouTube, StackOverflow, and LLM's (machine learning powered large language models, like ChatGPT). Here is what we believe you should use them for:

6.1 Youtube - Specialization

Youtube is full of free, complete tutorials introducing you to using new languages and libraries. We link to many videos throughout this guide that may be part of more complete and in depth tutorial series. If you find a certain library or framework interesting, have conducted brief research to determine it would be a good fit for your product, either by making you more efficient or improving performance, I recommend looking at Youtube first to learn how to use it. While the most popular videos are usually vetted, trusted resources, even videos with a smaller number of views can prove to be effective. Furthermore, these will often walk you through example projects, similar to this guide, which enforces the best teaching method of learning through doing.

6.2 Stack Overflow - Specific Coding Issues

As you grow your abilities as a developer, and begin to use more complex logic to implement certain functionality, you will run into some issues, either because you don't know how, or the best way to do something. You also may think you're doing something correctly, but for whatever reason, the functionality isn't what you expect. Luckily, many developers are generous with their time, and love to suggest possible solutions to your problems. Search Stack Overflow with your specific problem, and you'll likely find pre-written code, and/or advice, on how to perform the task at hand.

Stack Overflow is not the best for a concept overview, but rather suited for when you're already well versed in a certain technology, and are seeking the best way to use that tech for your product.

6.3 LLM's - Coding Tutor

In the past few years, AI development has been exponential. The functionality of LLM's are improving everyday, giving you the ability to converse with it in real time to discuss and solve problems. While people speculate it will replace all software development jobs, we live in an ideal time where it is not yet better than humans in every regard, but can seriously improve the performance and output of humans. However, a big barrier for new developers is that they don't know how to utilize the full functionality of the service.

Popular LLM's for code generation include Copilot, created by Github and powered on Open AI's gpt, which is super helpful because you can use it directly within VS Code. Another highly regarded service is Mixtral-8x7b, created by MISTRAL AI, which is performing better than most other LLM's in many regards. More info here:

<https://medium.com/thereach-ai/ai-showdown>

For beginners, I recommend sticking with the mainstream ChatGPT, created by OpenAI. It is the most popular LLM, accessible directly within your browser, and chances are you have already used the free version for other tasks.

I highly recommend paying the \$20 monthly subscription for GPT 4, as opposed to using the free version GPT 3.5. It has a serious upgrade in performance and you'll never be in doubt if you're getting access to the best information.

To see how we can use it, let's ask ChatGPT what it is capable of doing:

Me: What are the best ways to use an LLM to teach you how to code a new product?

ChatGPT: Using a large language model (LLM) like ChatGPT to learn coding for a new product can be highly effective. Here are some of the best ways to leverage an LLM for this purpose:

1. **Understand Fundamentals**: Start by using the LLM to grasp basic programming concepts and language-specific syntax. You can ask the model to explain concepts in detail or provide examples and comparisons to enhance your understanding.
2. **Interactive Learning**: Engage in interactive coding sessions where you present problems and the LLM provides step-by-step guidance on how to solve them. This can include writing actual code snippets, explaining the logic behind them, and troubleshooting errors in your code.
3. **Project-Based Learning**: Outline a project idea related to the product you want to develop, and use the LLM to help you break down the project into manageable steps. The model can help you plan the software architecture, select the right tools and libraries, and guide you through the implementation.
4. **Code Reviews**: Write code and have the LLM review it. The model can offer suggestions for optimization, point out potential bugs, and ensure your code adheres to best practices in coding standards.
5. **Debugging Assistance**: When you encounter bugs or issues, describe them to the LLM. It can help diagnose the problem and suggest fixes. This is an excellent way to learn how to troubleshoot and solve programming issues independently.
6. **Stay Updated**: Ask the model about the latest trends and best practices in software development relevant to your product. This can include new programming techniques, frameworks, and tools that could be beneficial for your project.
7. **Scalability and Maintenance**: Learn about practices for making your code scalable and easy to maintain. The LLM can provide insights into writing clean code, refactoring, and documentation practices.

8. **Soft Skills**: Coding a new product isn't just about technical skills. The LLM can also help you improve project management, collaboration, and communication skills that are crucial when working on coding projects, especially in team settings.

9. **Custom Examples and Explanations**: Tailor your questions to get customized examples or explanations that fit exactly what you need to learn or solve. This personalized approach can accelerate your learning process.

10. **Code Documentation**: Learn how to properly document your code. The model can guide you on writing effective comments and documentation, which is essential for long-term maintenance and team collaboration.

By integrating these approaches, you can effectively use an LLM to not only learn how to code but also manage and refine the development process of a new product

In my experience, ChatGPT was extremely helpful for concept overview, giving me a brief introduction into certain technologies before I decided to delve into it. Furthermore, it was great at laying out step-by-step plans of how I would add a specific feature into my code. It can also offer example code on how you would write something. Lastly, I found it super helpful for debugging. While it still cannot directly interpret files you upload, you can copy and paste large chunks of your code, and it will recommend how to debug, optimize or even fix it for you.

The best part of using AI is that it essentially is a 24/7 access expert tutor for any problem you may have. One of the main issues before learning how to code is first, you need to figure out what you need to learn. Then, you have to scour the internet for places to learn. An LLM expedites that process significantly, as you can ask it any question, small or large, and have it instantly output an answer. Using it to its fullest capabilities is essential for competing nowadays as everyone is utilizing it, and is a major driver of efficient coding.

Note: While AI generated code is very helpful, you have to be very cautious when using it directly. These models are trained on code it scrapes from the internet, outputting code based on what it acquired. This could be a security threat, if it is using out of date technology, or known vulnerabilities others could exploit.

Furthermore, it isn't always logical. While its output may give you the functionality you want (and not always), it may not be the best or most efficient way to do something. This is why it's better to use it for generating small code chunks for singular functions, as opposed to large processes. You always want to review the code against your own knowledge, as you'll likely be able to improve it.

Lastly, and perhaps most important to consider, because we are building an MVP for a product/service that we would like to eventually license or patent for a business, there can be serious copyright issues if you rely too heavily on AI Code. Again, due to the way it is trained, anything you utilize that is output by the model cannot be considered original work. When the code contains a mixture of human and AI generated work, such as a human prompting the model in a certain direction, and then improving upon the code, there is a large gray area in what can be used under copyright law. While it is still likely to be considered original work, the legality of this situation is still evolving, so be sure to constantly update yourself on it as you are using it. For more information, see here: <https://builtin.com/artificial-intelligence/ai-copyright#>. Overall, just be cautious to the extent you use ChatGPT, and check in with the model itself if you can use any code it helped you with.

7. Technical Set Up and Planning

7.1 IDE's vs Text Editor

When selecting the environment to start developing your code, your choices include an IDE (integrated development environment) or a text editor. Each of them has certain advantages.

Text editors are much simpler and beginner friendly. They won't consume a lot of computational resources, so you don't need a robust computer for developing, any modern laptop should suffice. They will help you manage your files, give syntax suggestions/completions, and color coordinate different functions and commands to help differentiate.

Common text editors include Sublime Text (<https://www.sublimetext.com/>), known for its customizability and speed, which offers a free trial, after which requires a \$70 license. Another is Atom (<https://atom-editor.cc/>), a free service that while not as fast as Sublime, seems to offer more customizable options, and powerful git integration (being developed by Github). Both are compatible with MacOS, Windows and Linux.

IDE's are more complex, and require more computational resources, but offer extensive additional features to experienced developers who can utilize its features. They will have a built-in text editor, but include additional tools like debuggers, AI tools for code completion, and code refactoring, which reorganizes your code to be more logical and efficient without changing its functionality. While IDE's typically support a wide variety of languages, they usually specialize in a specific one.

Popular IDE's include PyCharm (<https://www.jetbrains.com/pycharm/>), which specializes in Python, and IntelliJ IDEA (<https://www.jetbrains.com/idea/?var=1>), which specializes in Java.

Both are compatible with MacOS, Windows and Linux, and offer extensive plug-in options for additional functionality. Both were developed by the company JetBrain, and while they offer free licenses for students, are going to cost a significant amount for professional developers, anywhere from \$200 - \$800 depending on the variety your project requires.

For our purposes, we recommend VSCode (<https://code.visualstudio.com/>), a complex text editor released very recently by Microsoft, and is essentially a subset of the Visual Studio IDE (<https://visualstudio.microsoft.com/>). It is simple and lightweight like most text editors, but still offers lots of extensions and plugins, giving it similar functionality to IDE's as the developer sees fit. One of these is Live Server, which is crucial and we will touch on momentarily. It is free to use, and also offers git integration, to keep track of what files/versions have been saved. The editor itself was developed using Node.js (which we will be using) making it very compatible for front end languages. Most importantly for new developers, it is among the most popular development environments, and only growing in popularity. Many online guides and youtube tutorials will use VSCode, which makes it significantly easier to follow along.

To download VSCode and the Live Server, access the link above, and optionally follow the steps in this video from LearnCode.academy from 1:20 - 5:00 (https://www.youtube.com/watch?v=gQojMIhELvM&list=PLoYCgNOIyGAB_8_iq1cL8MVeun7cB6eNc&index=1&t=2s). It will walk you through how to install VSCode and LiveServer, a brief explanation of the user interface, and the steps of creating your first HTML file to test Live Server functionality.

Sources we used if you want to continue your learning on IDE's and text editors:

Text Editors vs IDE's - <https://www.linkedin.com/pulse/text-editors-vs-ides>

Comparing text editors - <https://codementorio.medium.com/best-text-editor>

VS Code vs Visual Studio - <https://www.freecodecamp.org/news/visual-studio>

Why VS Code - <https://code.visualstudio.com/docs/editor/whyscode>

7.2 Google Workspace

This step is optional, and if you have no idea what your business model or name is yet, and/or if you plan on running your business individually, this may not be necessary.

However, [Google Workspace](#) has proven to be a very effective tool for managing your business and conversing with coworkers. It is scalable, cheap ([pricing](#)), very secure, has reliable customer service, and is releasing new tools to increase productivity regularly. For more information on the potential benefits for your business, see here:

<https://gcloud.devoteam.com/blog/10-benefits-of-google-workspace>

The reason we recommend setting this up ahead of time, is as we start to use additional services and tools, it will often require an email setup. While you can use your personal email, should your business find traction, your data will be spread out across multiple accounts and hard

to manage. It's best to have a central location from the start to avoid mismanagement of data in the future.

Lastly, it will make setting up custom MX, mail exchange records, extremely easier once we get into DNS and hosting, as many hosting platforms have built-in Google Workspace functionality. It will also allow you to have a custom business email address, greatly increasing your perceived credibility, such as mikey@college-athlete-connect.com. You will have to acquire a domain first to do this, as we cover in section 11.1, so skip to there temporarily if you plan on using workspace.

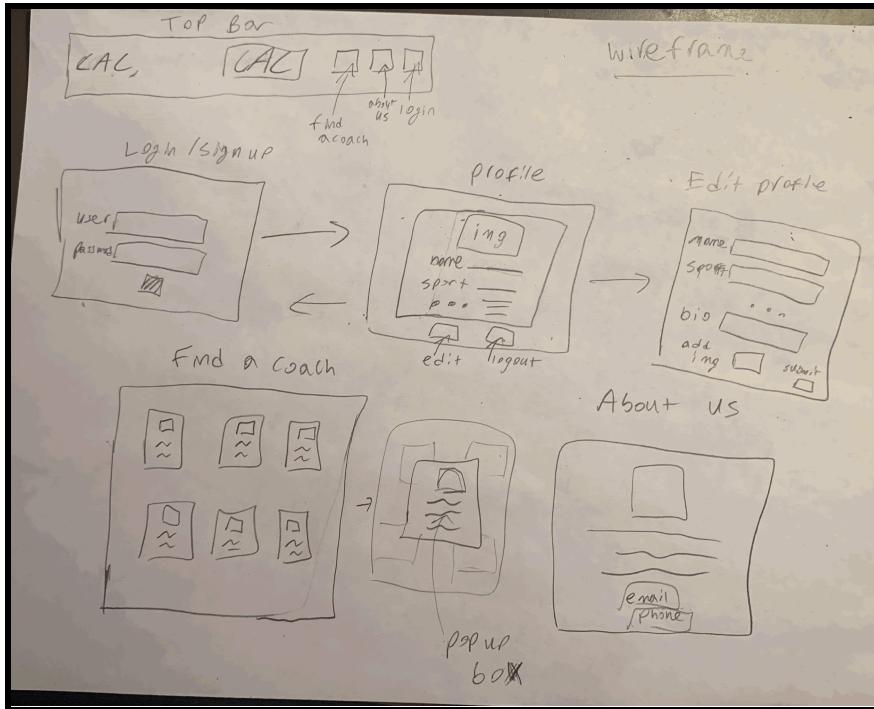
Here is a video to help you set up Google Workspace, up to 3:49 <https://www.youtube.com/watch?v=5BvxtZ4yQao>. If you don't have a custom domain yet, you can just skip this step for now within Google. However, if you already have one, continue with the steps in section 11.3.

7.3 Wireframing

A wireframe is essentially the “skeleton of your app, website, or other final product. Your wireframe shows the design team and stakeholders the bare-bones outlines of essential webpages, components, and features, including: Screen layouts, Navigation bars, Components of UX and UI design, and Interactive elements”. For more information on wireframing, see: <https://www.figma.com/resource-library/what-is-wireframing/>

You may be inclined to skip this step and jump straight into developing, attempting to figure it out as you go... but don't just yet! While it is almost sure your website layout will change and adapt as you progress, it is still a good idea to have a rough idea of everything you will need before you start developing. As we get into the concept of linking, which is connecting different pages in your website, so you can access it from another page (like a navigation bar), having a wireframe will keep your development space much more organized and clean, and prevent you from having to go back and alter a lot of code due to bad planning. It's also a good idea to draw arrows between web pages you want to be connected to for this reason.

To create a wireframe, you can either sketch it manually on paper, or design it digitally. If functionality is your main concern, I recommend using paper. It is quicker and offers a lot more flexibility. For example, this is one of the original wireframes of College Athlete Connect:



If design is your main concern, I recommend using a digital tool. The most commonly used tool is [Figma](#). Figma allows you to design high quality, detailed layouts of pages and the user experience. Furthermore, there are also some plugins available that allow you to convert your Figma design into HTML and CSS code, such as this one:

<https://www.figma.com/community/plugin>

These plugins typically don't output high quality code, at least in my experience, it tends to be convoluted and unorganized. However, it could be a good method to see what your code structure may vaguely look like, especially when just starting.

7.4 Branding Tools

The development process is significantly easier if the creative aspects of your business have been decided beforehand. These are what I recommend you test and complete before developing:

Name - Select a name for your business. This is the name people will see when they visit your website. Be wary that if you want a specific domain for your website it may already be taken. To check if a domain is available, you can use this link:

<https://www.namecheap.com/domains/registration/>

If your name is taken, consider using dashes between words or slight variations such that your name is still recognizable. You can also hire a domain agent to make an offer for a specific domain if you see it as crucial.

Logo and Color Scheme - Design a logo, and pick a color scheme for your brand image. More experienced graphic designers may opt to do this using a tool like Gimp or Adobe Photoshop. If you are inexperienced in design like myself, I recommend using [Canva](#). They offer plenty of templates to get you started, with a simple environment that you can easily add text, shapes and images to. Furthermore, they offer brand templates. This allows you to keep track of your specific color schemes, fonts, graphics and other components to keep everything consistent. Having a predefined color scheme will make CSS coding decisions much easier.

Branding Images - When you are structuring your website, it may be difficult if you don't know what images you're using or where you're going to put them. Styling is also easier when you know the image size before you implement it. For this reason, try to collect any additional photos you want to use and crop them accordingly.

7.5 Software and Accounts

Before we get into actual coding, it's best if you set up accounts and download some of the necessary software ahead of time. Here is the rest of what you'll need:

Github account - Github is a service that specializes in version control, which is saving different versions of your code through different stages of development. It is similar to [Dropbox](#) or [Google Drive](#). While this version control through Github is primarily meant for code, it can be used for other types of files as well. We will discuss this more later, but you can sign up for an account with your business email here: <https://github.com/>. For a more thorough description of what Github is and using the interface, see here:

<https://kinsta.com/knowledgebase/what-is-github/>

Git - To use Github, you'll need to download the git software to your computer. Use this link to download and follow the instructions it walks you through: <https://git-scm.com/downloads>

Node.js and npm - Node.js is a runtime environment that comes with npm, node package manager. A runtime environment is essentially just the space/software that allows your code to run. Instead of JavaScript just running in your browser, it can now run on your computer. It is not vitally important you understand the details behind what it is and how it works, but if you want to familiarize yourself more, see here: <https://www.techopedia.com/definition>.

Node.js is super useful, as it will allow us to use JavaScript on both the frontend and backend of our application (we discuss what this means later in the guide), significantly cutting down the amount of information we need to learn. To download Node.js, use this link and follow the steps: <https://nodejs.org/en>

Terminal/Command Line - The command line interface, with corresponding desktop app Command Line on windows, and Terminal on Mac, is a more direct way that developers use to access and change their file directories, as opposed to your File Manager/Finder. You can also execute commands to download or send certain files. We discuss this more, but familiarize yourself with where it is in your computer, and pin it to a location that's easily accessible, as we will be using it frequently.

7.6 Browsers

While you can most likely just use the default browser on your device, here are some you may want to consider getting:

Google Chrome - Being the most popular browser, most of your users will use it as well, so you can gear your development perspective directly towards them. It is constantly updating its developer tools, and allows you to live-edit your code. It also runs on V8 JavaScript engine, which uses the just-in-time (JIT) compiler, making for fast performance of your code. This is our recommend browser: <https://www.google.com/chrome/>

Firefox - You can get the regular or developer edition, which will have additional tools within the console, and extensive color-coding for easy debugging, particularly of CSS. It is also known for highly valuing privacy, so if you're developing an app with sensitive information, you can be sure it's protected: <https://www.mozilla.org/en-US/firefox/developer/>

Safari - The default browser on Mac computers, it has a powerful inspector tool, and is very efficient in using Mac resources for large applications

Microsoft Edge - The default browser on Windows computers, it is compatible with most Chrome extensions, and has built in developer tools for developer tools

8. Coding: Where to Start

Now that you have the full context and setup, we are ready to start actually writing code. You'll likely have to refer back to this section continually as you write code, but it's best to read through first, and get a high level understanding, before diving into the specifics.

8.1 HTML and CSS

In sum, HTML is the skeleton of all webpages, and CSS is the flesh. HTML tells your webpages how things should be laid out, and CSS tells it how things should look.

To give you a brief introduction into HTML, I recommend watching this video first. It will tell you what HTML is used for, what it's used with, and an introduction into what tags are and different types: https://www.youtube.com/watch?v=salY_Sm6mv4

Now that you understand what HTML is, we can start coding directly in our VS Code environment. The videos below will walk you along example projects. You may be inclined, for efficiency, to just watch the videos and then attempt your own code after. However, I STRONGLY recommend you follow along, input the code yourself, and test different things. This will give you a much better understanding and make future steps significantly easier. They are all done in VS Code, so it's easy to follow along.

If you prefer a more granular, detail oriented approach, I recommend the following video: <https://www.youtube.com/watch?v=HGTJBPNC-Gw>. While long, it will walk you through many different aspects of HTML and CSS, in a slower, guided fashion.

Instead, if you're like myself, you may prefer to learn by attempting/testing things on your own. If this is you, I recommend the following videos from LearnCode.Academy. He shows you at a high level what to expect, leaves room for you to stray off and attempt things on your own, and is very quick and efficient in his teaching methods. He also utilizes Live Server, so you can directly see the results of what you are coding. Of all the tutorials I surveyed, I found these to be the best ones:

HTML Basics, Tags, Images and Structure:

<https://www.youtube.com/watch?v=RjHflb-QgVc&list>

Integrating CSS Styling, Fonts and Coloring:

<https://www.youtube.com/watch?v=J35jug1uHzE&list>

Semantic Tags, Layout, Sizing and Structure:

https://www.youtube.com/watch?v=dMK_3lH1YPo&list

Flexbox and Navigation Bar:

<https://www.youtube.com/watch?v=aRMIdKRYg6c&list>

Note: Flexbox is CRUCIAL to efficient web development, and making your platform compatible across devices with minimal extra work. I highly recommend you familiarize yourself with it. While it may seem easier in the short term to neglect it, and structure your

layout using solely pixel sizes and percentages, you will completely have to redevelop your app for mobile.

As you grow more comfortable developing, for a more in depth coverage on Flexbox, I recommend the following video: <https://www.youtube.com/watch?v=3YW65K6LcIA&t=0s>

You may also find the following reference guide helpful with prewritten code:
<https://flexbox.malven.co/>

For other, more complex structuring methods using CSS, you can also refer to the following videos:

CSS Grid: <https://www.youtube.com/watch?v=EiNiSFIPIQE>

CSS Positions: https://www.youtube.com/watch?v=MxEtxo_AaZ4

Furthermore, if you believe your web app will be primarily used on mobile devices, you may want to consider incorporating responsive design (responding to different screen sizes), and developing mobile first. This video will cover the basics of responsive design:

<https://www.youtube.com/watch?v=fgOO9YUFlGI&list>

Lastly, another interesting CSS method is modal, which creates a popup on top of your current page, and is used on the Find A Coach Page on CAC, more information on it here:

https://www.w3schools.com/howto/howto_css_modals.asp

Multiple Pages and Linking:

<https://www.youtube.com/watch?v=iXSSHlOe47s&list>

CSS Selectors, Class Names, Ids and Pseudoclasses:

<https://www.youtube.com/watch?v=dcCCOiQ1ZuM&list>

Note: As mentioned in a previous video, there has been a movement away from div tags into using more specific semantic tags (that function the same as div tags, but make code much more readable). You will find these particularly useful when styling your page. For more information on different semantic tags, and why and how to use them, refer to this article:
<https://dev.to/kenbellows/stop-using-so-many-divs-an-intro-to-semantic-html-3i9i>

Best Methods for Incorporating Images:

<https://www.youtube.com/watch?v=7cwRaTqR4k0&list>

The above videos should get you up to speed on all HTML and CSS methods you'll need to start developing. As you branch off into your own development, you may also find these reference guides useful:

HTML Attributes: https://www.w3schools.com/tags/ref_attributes.asp

CSS Properties: <https://www.w3schools.com/cssref/index.php>

Hint: Use Command or Control + F to quickly navigate through the guide

Also, not all attributes/properties will be usable on all modern browsers. To see if a particular tool you want to use will appear correctly on different browsers, use the following website, which will tell you what versions its usable on: <https://caniuse.com/>

8.2 Git and Version Control

Now that you have git downloaded, a Github account setup, have a basic understanding of what it's used for, and a few example projects saved, you'll want to learn how to effectively use version control through the terminal.

Desktop Version - Before we start, it should be mentioned that Github does offer a desktop version that allows you to bypass the terminal: <https://desktop.github.com/>. For a tutorial on how to use it, see here: https://www.youtube.com/watch?v=v_1lqtOnUMg

However, this should really only be used if you severely want to avoid using the terminal. It is actually quite simple and easier to learn, and you'll need to use it for other aspects later in this guide, so I recommend following along with the rest of this section.

Terminal and Repository - For a full overview of using the terminal with Github, refer to this video: <https://www.youtube.com/watch?v=0fKg7e37bQE&t=390s>

While it is a bit older, so the github user interface may not be the exact same, of countless videos I've scanned, it's the only one that really covers this at a beginner level.

The first 4:30 covers why it is useful when working in a team. Even if developing alone, it still adds to your understanding.

After that, you are walked through the steps of creating a repository (a storage space for versions of your code), how to clone that repository into your computer, and use common commands like cd, ls, push, add, commit and pull to navigate through directories and update your code accordingly.

For practice, try creating a repository for your practice code you just created, make some changes, and see if those changes are visible within Github. If it is your first time using git, you will likely have to generate a personal access token within Github, and use that as part of a command in the terminal to authenticate yourself and give your computer access to the repository. Here is a video covering the process:

<https://www.youtube.com/watch?v=0fKg7e37bQE&t=390s>

And here is the command you'll need:

```
git remote set-url origin https://<your token>@github.com/<your username>/<repository name>
```

When filling in the fields, remove the <> as well.

This code will only ever be shown once, so you may want to copy it into a separate secure text file or write it down for future use.

Note: I recommend against cloning a repository into the cloud. These are two different version control systems, and may interfere with each other and lead to unexpected changes. Instead, save it locally, directly on your computer, within a subdirectory in your Documents folder.

Reference - No matter how often you use the terminal, you may find yourself forgetting common commands. All developers do this. Refer to this guide directly from Github as a reference of commonly used commands for git:

<https://training.github.com/downloads/github-git-cheat-sheet.pdf>

Github in VS Code - VS Code is very convenient in that it comes with out of the box Github integration. It will highlight your code to show what has been added and deleted, and allows you to push, pull and merge branches, all from directly within the VS Code interface. I still recommend using the terminal before switching to this method, so you actually understand what is going on under the hood. However, you may find it more convenient for your development, especially if only working with one monitor. To see how to use this functionality, refer here:

https://www.youtube.com/watch?v=i_23KUAEtUM

Customizing Terminal - This step is by no means necessary, however it's no secret the terminal is not appealing to look at and likely one of the main deterrents from using it. If you spend a lot of time using the terminal, you may want to customize its color scheme and layout. To do so, refer to this video: <https://www.youtube.com/watch?v=CF1tMjvHDRA>. It will also give you some good practice downloading common packages and plugins like Homebrew and Zsh (Zshell).

8.3 JavaScript

The JavaScript language is by far the most difficult part to learn throughout this development process. It is an extremely versatile language and will drive the functionality of our completed web application. However, there is a lot to memorize and understand.

At this point, it's important to keep in mind that we are developing an MVP. We are looking for efficiency. For HTML and CSS, it is easier to understand the bulk of it up front, as there isn't much to learn, so you can quickly complete the design of your web pages. JavaScript

is unlike those, and to completely understand it and get comfortable with it is predicted to take anywhere from 3 - 6 months. For this reason, we recommend having a brief understanding of some of the basic functionality, and how to write/structure it. This will give you enough to start. After that, you only need to learn as you go. It would be quicker and easier to look up specific functions as you need them, as opposed to learning them upfront.

Below, we will walk you through how to gain this basic understanding, and some common functions you will use. We will cover more specific examples in the backend.

Alternatives to JavaScript - In this guide, we will be using JavaScript on both the frontend and backend (with Node.js). However, there are some other alternatives you may want to consider, based on your previous experience.

Frontend - Two popular alternatives to JavaScript include:

TypeScript - This is built on JavaScript, but adds additional syntax for types. It does this by adding static types, which is especially helpful with large codebases, as it adds additional checks for errors. It runs anywhere JS does, and comes with additional features like autocompletion. For more information, see here: <https://www.typescriptlang.org/>

Coffeescript - With syntax inspired by python (and good to use if you have python experience), this brands itself as “just JavaScript” as it compiles into a one-to-one equivalent. It is meant to highlight the best parts of JS, through a much simpler syntax, that increases readability and is easier/more efficient to write. For more information, see here: <https://coffeescript.org/#overview>

Backend - For a complete overview of all languages that compile to JavaScript, see the following list:

<https://github.com/jashkenas/coffeescript/wiki/list-of-languages-that-compile-to-js>. If you see any languages you may have had some prior experience with, or exposure to, you may want to refer to the guide on how to integrate it into your code and translate it to JavaScript when compiling. Some popular alternatives include:

Python - This is known to be very conducive to beginners due to its simplicity and easy readability. The syntax is extremely straightforward. It is high performance and great for handling server side logic. There are a lot of online resources explaining the best way to utilize it, as well. You may want to consider using it with a framework like [Django](#) or [Flask](#). For more information, see here: <https://www.python.org/>

Java - This is a class based and object oriented language, which significantly simplifies the logic for complex projects. It is great at handling memory and has an automatic garbage

collector, to prevent memory leaks and make it more secure. It can be used on any platform with the use of Java Virtual Machine (JVM). It also has a large number of online support resources and existing libraries to import. It is commonly used with [Spring](#) and [Java EE](#). For more information, see here: <https://www.java.com/en/>

Ruby - This language is known for having a concise syntax, and is almost always used with the [Rails](#) framework. It values “convention over configuration” by providing default structures for pages and databases, assuming what the developer will need, unless otherwise specified. This makes for rapid development, and allows applications to scale with the user's knowledge as they learn more. For more information, see here: <https://www.ruby-lang.org/en/>

Introduction to JavaScript - As you start, I recommend watching this quick video: https://www.youtube.com/watch?v=c-I5S_zTwAc. It will be hard to understand now, but will give you a brief overview of what we will be learning, including what it is/how it's used, frameworks, functions/methods, and asynchronous programming. JavaScript is the single most popular programming language, and its versatility proves why it is.

If you are completely new to programming, and have enough time to learn in depth, I recommend watching and following along a longer course. This 12 hour course was very helpful for me and works on an example project emulating Amazon (a horizontal marketplace): <https://www.youtube.com/watch?v=SBmSRK3feww&t=0s>

Instead, for a simplistic but efficient approach, I once again recommend a LearnCode.academy youtube series, this time on JavaScript. Nearly all other tutorials are extremely long, but seem to still miss smaller details this covers. It should give you just enough to get started:

Using the Console, Variables and Types, Prompts, Alerts, Assigning Values:
<https://www.youtube.com/watch?v=fGdd9qNwQdQ&list>

Comparison Operators, If Else Statements:
<https://www.youtube.com/watch?v=Gy87ZS5sb1w&list>

Functions, Script Tags in HTML, Arguments, Return and Undefined values:
https://www.youtube.com/watch?v=AY6X5jZZ_JE&list

Arrays, While and For Loops:
<https://www.youtube.com/watch?v=orAS-MBh5f4&list>

Document Object Model (DOM), Elements, Query Selector, Parents and Children:
<https://www.youtube.com/watch?v=W3EK4MIZW4g&list>

Events, Event Listener, “this” Keyword, Selectors, Attributes:
<https://www.youtube.com/watch?v=e57ReoUn6kM&list>

“This” Keyword, Span, Handling Input, Keyboard Events
<https://www.youtube.com/watch?v=0tEW8rB1bbU&list>

Commonly Used Keywords - Given that the above tutorial is older, and JS is constantly evolving, it doesn’t cover everything you should know. Here are 3 commonly used keywords that will come up frequently you should familiarize yourself with before starting (more info on them in the reference guides above):

Import - In order to bring in modules, either from within your own code base, or an external source on the internet, you’ll need to use an import statement. It is written at the top of your script tag, as “`import { module 1, module 2, ... } from './file path'`” or “`import { module 1, module 2, ... } from 'url'`” where the url is the location of a published module on the internet. This will be crucial for our backend, and can save a significant amount of work if someone has already written code for a solution.

Async - This comes before a function to declare it as an asynchronous function, meaning it will return a promise (see definition below). This means the promise does not need to be fulfilled in order for the rest of the code to execute. You can also add an await keyword within asynchronous functions, which temporarily stops the execution of just that individual function until the assigned promise is completed.

Promise - In JavaScript, a Promise is an object that will produce a single value some time in the future. If the promise is successful, it will produce a resolved value, but if something goes wrong then it will produce a reason why the promise failed. JavaScript promises can be in one of three possible states. These states indicate the progress of the promise. They are:

- *pending*: This is the default state of a defined promise
- *fulfilled*: This is the state of a successful promise
- *rejected*: This is the state of a failed promise
(<https://www.freecodecamp.org/news/guide-to-javascript-promises>)

Fetch - This is a modern API meant to simplify HTTP requests, and allows you to do network requests that will load data from a server. It returns a promise for the response object that can be handled accordingly. It is typically good convention to write code to handle the case where the promise is fulfilled or rejected.

Reference Guides - Here are some reference guides you may find helpful as you branch off into your own JS development:

Javascript Events - <https://developer.mozilla.org/en-US/docs/Web/Events>

Keyboard Event Values - <https://css-tricks.com/snippets/JavaScript/JavaScript-keycodes/>

High Level Overview on Commonly Used JavaScript Features -
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>

In-depth JavaScript Reference Sheet -
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>

Frameworks - As you continue on your development journey, you may want to consider using a Javascript framework for the frontend. A framework is a set of prewritten code, tools and libraries meant to simplify development by using reusable components, as opposed to writing all individual code themselves.

For our use, I do not recommend using a framework just yet, until you have a grasp on vanilla JavaScript (plain with no additional tools). Frameworks may shorten syntax or conventions, and you won't fully understand what it is doing or how to use it if you don't know JavaScript first.

However, if you find yourself developing a platform that is centered around the user experience, large scale, and design is very important, you may want to consider using a framework to make the process more efficient. This video explains why you might want to use a framework and how to pick one: <https://www.youtube.com/watch?v=VbvMJUpY0a4>. This video also explains the logic behind how they work:
<https://www.youtube.com/watch?v=Ka77djMkSwg&t=326s>

The current most popular JS frameworks are React, Angular, and Vue. There will be a lot of online resources to learn and use these tools. A video explaining which one might be best for you can be found here: <https://www.youtube.com/watch?v=xDhzYQ4VyCw>

While I cannot recommend a specific framework as this is completely dependent on your individual needs, here is a quick overview of the. All of them see similar performance:

[React](#) - This is very popular with startups, and is focused purely on UI, so it has an extensive set of tools for the frontend. However, it is written using JSX, an extension of HTML, and while not necessary to use JSX, there is an additional learning curve. It also does not offer backend support, so you'll have to find and incorporate different libraries for this purpose.

[Angular](#) - This is meant to build full stack solutions, and while verbose, it may be easier for developers new to JavaScript to follow along. It has an extensive API, so there is a steep learning curve, but it will incorporate all the tools you need for a full stack solution.

[Vue](#) - This is a newer framework and won't have as much online support as the previous two. However, it offers more full stack solutions than React, but less than Angular, and is known to be simple and conducive to beginners who may be new to UI.

I recommend going through the backend solutions in this guide first before deciding which framework, if any, is the best fit for your web app.

8.4 Scalable Coding Practices

Before you start developing your own code, remember these tips to keep your code readable and easy to revise. The total lines of code you have written will increase quicker than expected, and it's easy for it to get messy and hard to track. This will make it significantly harder for you to scale, add new features, or bring new team members in to help write. While it may seem easier in the short term to write "bad" code, it will take twice as long for you to go back and clean it up, so it's best to do it right the first time.

Version Control - In the same way it's good practice to consistently save an essay you are writing, you also want to save your code. It is actually even more important in this case, as trying to implement a single feature may completely change or ruin the functionality of your website. Before every major change you make, I recommend pushing a version of your current code to the cloud. That way, if the feature change is unsuccessful, you can simply pull the old version of your code to get it back to working version. It is very common that you'll need to use old versions of your code, and if you save frequently, you will prevent yourself from rewriting things you've already successfully done.

On a side note, if you're working with a team, it's important to keep them updated on what you're working on. In the unlikely case there was bad communication and you're working on the same file, consistent version control will allow you to catch the mistake quickly and stop you from working on different versions of the same thing.

DRY Code - When writing, remember to use Do Not Repeat yourself code. If you find yourself writing the same thing multiple times, it's usually best to move that code into a separate file, and then use a single line to link to that code. This will happen very often during styling, but can also

occur with HTML (such as a navigation bar you use on every page) or with some JavaScript commands (usually when importing libraries).

If you have the same code in different locations, and want to change it (news flash - you're going to want to change it in the future), you'll have to find every instance and change each line individually. However, if it is in a separate file, you'll only need to change it once, and every feature that links to that code will update accordingly.

Efficient Use of Resources - When you're coding, make sure you're taking full use of your resources. This especially applies to LLM's. While it is super helpful to have reference guides and tutorials, if you ever find yourself “searching” for something, such as a particular function or library to implement a feature, it's usually best to just ask the LLM how to do it. It can also lay out a rough frame for your HTML and CSS, which you can then improve, as opposed to starting it from scratch, which is often the hardest part. It is also extremely well versed in JavaScript, and can give you a rough estimate of appropriate code for functions spanning many different libraries. Because JavaScript is significantly harder to learn, and you only need to use it to the point to get minimum functionality (unless you want to become an expert coder), an LLM can help you find or write the appropriate functions for intended functionality, in small individual pieces.

Separate Style Sheet - In your practice code, you probably wrote your style within the head of your HTML. As you work on multiple pages, which your web application will have, you want to keep a separate file containing your CSS. As you add styling to HTML, you can do a side-by-side view of your CSS file and HTML file. If your website has a consistent feel and look (which you want) you'll probably end up using the same style methods frequently. Before writing, you can quickly scan through your pre-written CSS to see if you've already coded for that instance. VS Code will also highlight any code you have written that is repeated when hovering over it.

Good Class Naming - This relates to the point above, but when adding sections and dividers to your webpage, make sure you are using unique, specific class naming to help you easily identify what each section does. This will help outside readers determine the functionality. Also, when applying styling, you can use commas to apply it to multiple classes at the same time, enforcing DRY code. An extension of this would also be descriptive, consistent commenting on your code, to help you remember what purpose a certain block or section serves.

Console Debugging - When viewing your webpage in a browser, you can right click to find “inspect element” which will bring up the console. Here, you can view any possible errors that were thrown when running your code, and find exactly where that error occurred. It is also good practice to regularly include “console.log(“__”)” throughout your JavaScript to mark specific

sections. This can make it easier to locate where specific errors are occurring, and see what's functioning correctly.

Script Location - Unless you specifically have a reason to include script tags earlier in your code, it's best to insert it at the bottom, right before the closing `</body>` tag in your HTML. While asynchronous loading will usually solve the problem, sometimes it will take your web page longer to load, or not load at all, if your JavaScript references specific elements in your HTML (because you haven't allowed those elements to load in yet). Even if it functions correctly, it could seriously hinder the user experience and make your website appear "slower" even if it's running at the same speed, just loading elements in a different order. You want the client slide to load first, so use script last when possible.

9. Front End Development

Throughout this section, we will give you suggestions and links to how to actually build your web app, and reiterate some common strategies you'll likely have to refer back to.

9.1 Organization

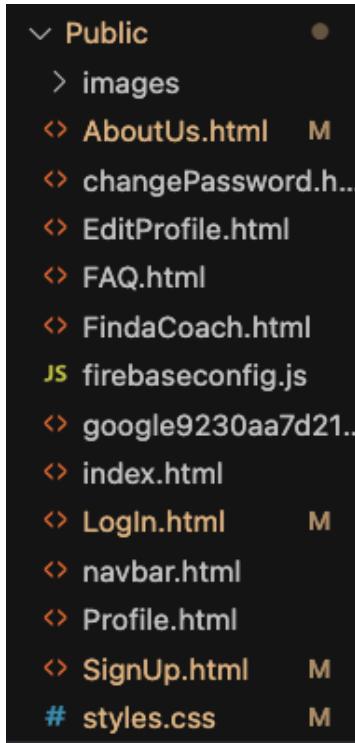
After you have completed wireframing, you should have a good idea of what webpages will be required for you to edit. Once we eventually get to the stage of deploying our website, we will need to have all of the files we are publishing in a single folder/directory. The common naming convention for this is '**public**' so it's best to start with that as your main folder.

After that, within **public**, create an empty html file for each of your web pages with a logical name. It's considered good practice to use camelCasing when naming your files (https://en.wikipedia.org/wiki/Camel_case). If you have greater than 10 files, you may want to consider subdirectories within your main folder. It's best to determine your directory layout ahead of time, as it makes linking (https://www.w3schools.com/html/html_links.asp) your webpages much easier. Otherwise, if you change it after (which you still can), you'll have to go back and edit the relative urls within each page.

Next, you'll want to have separate files for your CSS styling. A good naming convention for this is **main.css**. Our application is relatively small scale, so all of our styling will be in a single stylesheet. However, as your web app grows, you may want to split up your styling into separate sheets, so it's best to avoid names like "style.css".

Lastly, chances are you will want to include your logo or other branding images that you collected earlier, within the website, so create a file named "images" and upload all images you think you might need ahead of time.

Here is an example of the file directory for CAC:



9.2 Recommended Development Ordering

As you begin and continue to develop your web app, you'll become more comfortable and familiar with the coding languages you are using. Here is our recommended development ordering, scaled from easiest to most difficult, along with the corresponding web pages within our CAC example:

1. Text-Based, Non-Interactive Pages → FAQ, About Us, Home (index.html)
2. Header → Navigation Bar (up top)
3. Pages with input fields → Sign Up, Login, Edit Profile
4. Pages requiring backend data → Profile, Find a Coach

I also recommend you don't add any JavaScript yet, and wait until we get to the backend to add it all at once. Some pages may require script to add full functionality, so you may want to wait to develop those until we learn how to incorporate the backend.

9.3 Implementation

Every project will be different, so I can't tell you exactly how to code what you need. But based on everything we have learned so far, I will show you examples with CAC of how certain features were implemented, based on the ordering above. You can use this code to see how specific features were implemented, to assist you in your own development.

1. Text-Based Page

Example: About Us Page

College Athlete Connect  Find a Coach | FAQ | About Us | Profile

Made by Athletes, for Athletes.



Founder: Mikey Squires, Princeton Wrestling 174lbs, COS BSE c/o 2025

College Athlete Connect: Our Story

College Athlete Connect was founded on the belief that every high school athlete deserves the opportunity to continue their sport in college. For most aspiring athletes, the problem is not a willingness to do the work, but finding the right coaches to bring them to the next level.

Growing up in a rural town in Upstate NY, I was hard-pressed finding wrestling coaches to elevate me to the level of a D1 recruit. Our only solution was my father leaving straight from work, driving me an hour in each direction to practice four nights a week, depriving me of time with my mother and sister at home. Many of my teammates were forced to do the same thing. With the countless sacrifices already made to be a high level athlete, this is not one that should have to be made.

Growing up in a rural town in Upstate NY, I was hard-pressed finding wrestling coaches to elevate me to the level of a D1 recruit. Our only solution was my father leaving straight from work, driving me an hour in each direction to practice four nights a week, depriving me of time with my mother and sister at home. Many of my teammates were forced to do the same thing. With the countless sacrifices already made to be a high level athlete, this is not one that should have to be made.

Now that I am on the opposite side, a college student-athlete strapped for cash, I would love to give back to the community that gave so much to me. But for someone looking to coach, there is no way to find clients without handling the marketing yourself.

College Athlete Connect serves as a two way solution. Any athlete in need of a coach can find one. And any athlete looking to coach can find a client. We handle the rest. We will continue growing to serve the greatest number of athletes possible, and make collegiate athletics accessible to everyone.

CONTACT US

Please reach out for business inquiries, as well as questions/concerns about our business or platform.

Email: mikey@college-athlete-connect.com

Phone: (607) 373-9330

2024 College Athlete Connect

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About Us - College Athlete Connect</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <div id="navbar-placeholder"></div>
    </header>
    <script>
        fetch('navbar.html')
```

```

.then(response => response.text())
.then(text => document.getElementById('navbar-placeholder').innerHTML = text);
</script>

<main class="about-box">
  <section class="about-top">
    <h1>Made by Athletes, for Athletes.</h1>
    <figure>
      
      <figcaption><em>Founder: Mikey Squires, Princeton Wrestling 174lbs, COS  
BSE c/o 2025</em></figcaption>
    </figure>
  </section>

  <section class="about-info">
    <h1>College Athlete Connect: Our Story</h1>
    <p>
      College Athlete Connect was founded on the belief that every high school athlete deserves the opportunity to continue their sport in college. For most aspiring athletes, the problem is not a willingness to do the work, but finding the right coaches to bring them to the next level.
    </p>
    <p>
      Growing up in a rural town in Upstate NY, I was hard-pressed finding wrestling coaches to elevate me to the level of a D1 recruit. Our only solution was my father leaving straight from work, driving me an hour in each direction to practice four nights a week, depriving me of time with my mother and sister at home. Many of my teammates were forced to do the same thing. With the countless sacrifices already made to be a high level athlete, this is not one that should have to be made.
    </p>
    <p>
      Now that I am on the opposite side, a college student-athlete strapped for cash, I would love to give back to the community that gave so much to me. But for someone looking to coach, there is no way to find clients without handling the marketing yourself.
    </p>
    <p>
      College Athlete Connect serves as a two way solution. Any athlete in need of a coach can find one. And any athlete looking to coach can find a client. We handle the rest. We will continue growing to serve the greatest number of athletes possible, and make collegiate athletics accessible to everyone.
    </p>
  </section>
</main>

```

```

        </p>
    </section>

    <section class="contact-info">
        <h2>Contact Us</h2>
        <p>Please reach out for business inquiries, as well as questions/concerns about our business or platform. </p>
        <p><a href="mailto:mikey@college-athlete-connect.com">Email: mikey@college-athlete-connect.com</a></p>
        <p><a href="tel:+16073739330">Phone: (607) 373-9330</a></p>
    </section>
</main>

<footer>
    <p> 2024 College Athlete Connect</p>
</footer>
</body>
</html>

```

The stylesheet was linked by:

```
<link rel="stylesheet" href="styles.css">
```

The navbar was imported using the fetch API we mentioned:

```

<header>
    <div id="navbar-placeholder"></div>
</header>
<script>
    fetch('navbar.html')
        .then(response => response.text())
        .then(text => document.getElementById('navbar-placeholder').innerHTML = text);
</script>

```

Clickable links that forward you directly to your devices phone or email were implemented using href

```
<p><a href="mailto:mikey@college-athlete-connect.com">Email: mikey@college-athlete-connect.com</a></p>
<p><a href="tel:+16073739330">Phone: (607) 373-9330</a></p>
```

Furthermore, the styling used to center the image, have different text sizes, and make the links appear as clickable buttons, were implemented as such:

```
/* About Us */
```

```
.about-top, .about-box, .about-info, .contact-info {
    padding: 0 30px;
}
.about-box {
    margin: 0px auto;
    max-width: 1200px;
}
.about-top h1 {
    font-size: 32px;
    font-weight: bold;
    font-style: italic;
    color: #000000;
    text-align: center;
    margin: 0;
    margin-top: 0;
    padding-top: 20px;
}
.about-top img {
    display: block;
    margin: 2px auto 5px;
    width: 30%;
    border: 5px solid #ddd;
    box-shadow: 0 0 8px #aaa;
}
.about-top figcaption {
    text-align: center;
    margin-bottom: 30px;
}
.about-info h1 {
    text-align: center;
    font-size: 24px;
}
.contact-info h2, .about-info p, .contact-info p {
    text-align: center;
    font-size: 18px;
    line-height: 1.6;
    margin: 0 auto;
    padding: 0 15px;
    margin-bottom: 20px;
}
.contact-info h2 {
```

```

font-size: 24px;
color: #000000;
text-transform: uppercase;
margin-bottom: 10px;
}

.contact-info p {
    font-size: 18px;
    line-height: 1.6;
    margin-bottom: 15px;
}

.contact-info a {
    display: flexbox;
    background-color: rgb(255, 49, 49);
    color: #fff;
    padding: 10px 20px;
    text-decoration: none;
    border-radius: 5px;
    transition: background-color 0.3s ease;
}

.contact-info a:hover {
    background-color: #cc3d3d;
}

```

To make our email and phone number links appear as buttons, we added background coloring and padding around them, as such:

```

.contact-info a {
    display: flexbox;
    background-color: rgb(255, 49, 49);
    color: #fff;
    padding: 10px 20px;
}

```

Furthermore, to signify to the client that it is clickable, you can add a case for the hover event, such that the background slightly changes color

```

.contact-info a:hover {
    background-color: #cc3d3d;
}

```

2. Header

Example: Navigation Bar

The navigation bar is crucial to allowing your user to easily navigate the website, and it should appear on every page.



```
<nav class="nav-bar">
    <div class="left-links">
        <a href="index.html">College Athlete Connect</a>
    </div>
    <div class="nav-logo">
        <a href="index.html"></a>
    </div>
    <div class="right-links">
        <a href="FindaCoach.html">Find a Coach</a>
        <a href="FAQ.html">FAQ</a>
        <a href="AboutUs.html">About Us</a>
        <a href="Profile.html">Profile</a>
    </div>
</nav>
```

Note that it is divided into three sections, a left link, center image, and right links. This code was placed in a separate file to easily allow other pages to link to it. It used the following styling:

```
/* Navigation Bar */
.nav-bar {
    width: 100%;
    display: flex;
    justify-content: space-between;
    align-items: center;
}
.nav-bar > div {
    display: flex;
    align-items: center;
}
.left-links, .right-links {
    display: flex;
    align-items: center;
    list-style: none;
    margin: 0;
    padding: 0;
    flex: 1;
}
.left-links {
```

```
    justify-content: flex-start;
}

.right-links {
    justify-content: flex-end;
}

.left-links a {
    color: white;
    text-decoration: none;
    padding: 10px 15px;
    font-size: calc(1.5vh + 1vw);
    font-family: Arial, sans-serif;
    font-weight: bold;
    background-color: transparent;
    border-radius: 0;
    transition: color 0.3s;
}

.right-links a {
    color: white;
    text-decoration: none;
    padding: 10px 15px;
    margin: 0px 5px;
    font-size: calc(1vh + 1vw);
    background-color: rgba(0, 0, 0, 0.2);
    border-radius: 5px;
    transition: background-color 0.3s, transform 0.3s;
}

.right-links a:hover {
    background-color: rgba(0, 0, 0, 0.4);
    transform: scale(1.05);
}

.left-links a:hover {
    color: white;
    transform: scale(1.05);
    text-decoration: none;
}

.nav-logo img {
    max-height: 8vh;
    width: auto;
    text-align: center;
    flex-grow: 1;
}
```

To have the Navbar cover the entire width of the page, and have its elements evenly spaced, I used the following:

```
.nav-bar {  
    width: 100%;  
    display: flex;  
    justify-content: space-between;  
    align-items: center;
```

Using flex, each of the 3 sections have their own “box”. We made it so each element was centered within their own boxes, leaving empty space between them. We can also use flexbox within flexbox, which can be used to have the right links evenly spaced:

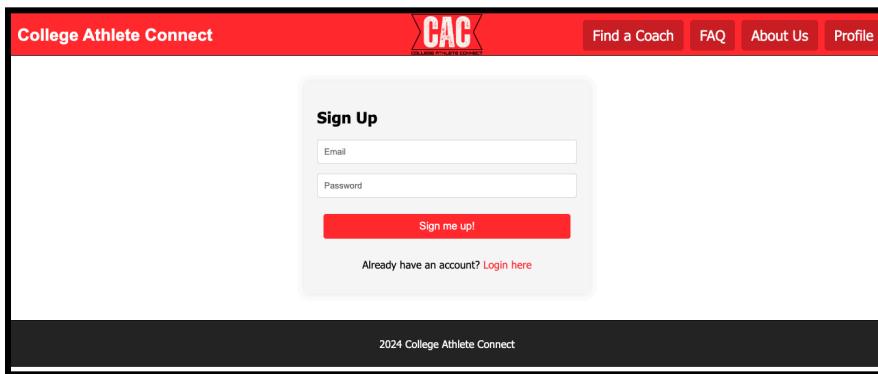
```
.left-links, .right-links {  
    display: flex;  
    align-items: center;
```

Another important note is that “vh” and “vw” were used frequently, usually in calculations. These stand for viewport height and width, and the number before it is a percent. So by setting the height of an image to 8vh, I always want it to take up 8% of the clients screen height. This can be used to keep font and image sizing consistent relative to the rest of the elements:

```
font-size: calc(1.5vh + 1vw);  
.nav-logo img {  
    max-height: 8vh;
```

3. Pages with Input Fields

Example: Sign Up



This was implemented with the following HTML:

```
<!DOCTYPE html>  
<html lang="en">  
<head>
```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Sign Up - College Athlete Connect</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <div id="navbar-placeholder"></div>
    </header>
    <script>
        fetch('navbar.html')
            .then(response => response.text())
            .then(text => document.getElementById('navbar-placeholder').innerHTML = text);
    </script>
    <main class="login-container">
        <h2>Sign Up</h2>
        <form id="signup-form" class="login-form">
            <input type="email" name="email" placeholder="Email" required>
            <input type="password" name="password" placeholder="Password" required>
            <button type="submit">Sign me up!</button>
        </form>
        <div id="signup-error"></div>
        <div class="signup-prompt">
            <p>Already have an account? <a href="LogIn.html">Login here</a></p>
        </div>
    </main>

    <footer>
        <p> 2024 College Athlete Connect</p>
    </footer>

    <script type="module">
        // Script covered later
    </script>
</body>
</html>

```

The ability to add an input for an email and password was done with the following:

```

<input type="email" name="email" placeholder="Email" required>
<input type="password" name="password" placeholder="Password" required>

```

- By adding a type of email, this requires the user input to have an “@” followed by a domain, otherwise an error message will be thrown telling the user to include it. The required keyword means the field must be filled in before the user can submit.
- You could also add something like “minlength= “7”” to the password field after required, if you want user passwords to have a minimum length of 7 (a good security measure)
- Placeholders, saying email and password, were also included, signifying to the user what to include in the field when it is empty

A button was also added by:

```
<button type="submit">Sign me up!</button>
```

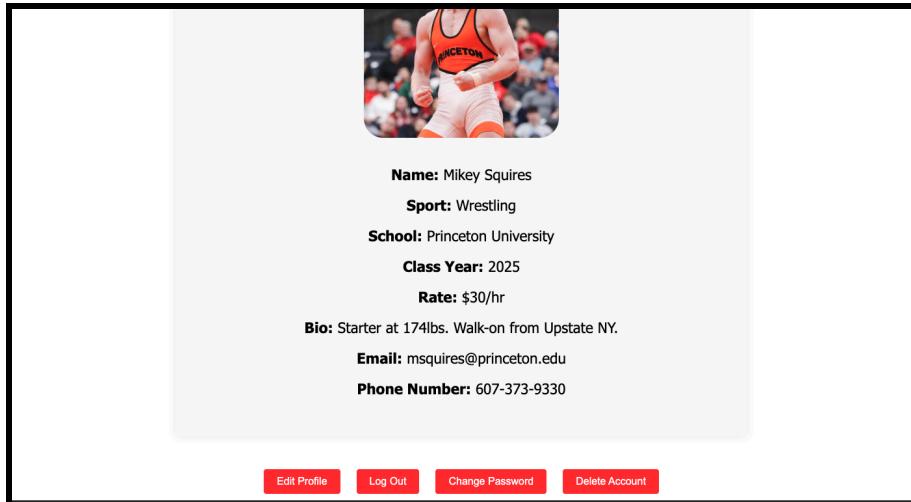
The type submit was not necessary, as this is the default setting, but it signifies it will be used to send form data to a server. By including it in the same form tag as the email and password inputs, this means we will be sending that data. We will cover more on script functionality later.

4. Pages Requiring Backend Data

Example: Profile

For these types of pages, I would wait until you start adding backend functionality, as the HTML necessary will seem insignificant alone. For example, see the Profile page:

Name: Mikey Squires
Sport: Wrestling
SCHOOL: Princeton University
Class Year: 2025
Rate: \$30/hr
Bio: Starter at 174lbs. Walk-on from Upstate NY.
Email: msquires@princeton.edu



It used the following HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Profile - College Athlete Connect</title>
    <link rel="stylesheet" href="styles.css">

</head>
<body>
    <header>
        <div id="navbar-placeholder"></div>
    </header>

    <script>
        fetch('navbar.html')
            .then(response => response.text())
            .then(text => document.getElementById('navbar-placeholder').innerHTML = text);
    </script>
    <div id="main-content">
        <main id="profile-container">
            <h2>User Profile</h2>
        </main>
    <div>
        <button id="edit-profile-btn">Edit Profile</button>
```

```

        <button id="logout-btn">Log Out</button>
        <button id="change-password-btn">Change Password</button>
        <button id="delete-account-btn">Delete Account</button>
    </div>

</div>
<footer>
    <p> 2024 College Athlete Connect</p>
</footer>

<script type="module">
    // Script covered later
</script>
</body>
</html>

```

All of the custom profile data will be populated within the User Profile:

```
<h2>User Profile</h2>
```

It utilizes the following CSS:

```

/* Profile */

#main-content {
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    width: 100%;
}

#profile-container {
    width: 70%;
    max-width: 800px;
    margin: 40px auto;
    padding: 40px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
    border-radius: 8px;
    background-color: #f9f9f9;
    text-align: center;
}

#profile-container img {
    width: 60%;
}

```

```

max-width: 300px;
height: auto;
border-radius: 10%;
margin-bottom: 20px;
}

#profile-container h2, #profile-container p {
    margin: 20px 0;
    font-size: 1.4em;
    font-weight: normal;
}

#profile-container p strong {
    font-weight: bold;
}

button {
    margin: 10px;
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    background-color: #ff3131;
    color: white;
    cursor: pointer;
    font-size: 1em;
}

button:hover {
    background-color: #ff6161;
}

```

It is evident that all of the CSS aspects and how it will populate is dependent on the data pulled from the server. Without it, it's extremely hard to test. For that reason, I recommend waiting until the backend section before creating these pages .

10. Back End Development

Now that you have most of your frontend designed, you can use your newly found JavaScript skills to add functionality to these pages. We will write all of our JS directly in script tags in HTML to talk to the server, but first, we need to use some tools that will help us do this. This includes Firebase, a BaaS, and Node.js, a runtime environment, which you may have already installed above.

10.1 BaaS Platforms

To develop our backend, we will be using a BaaS (Backend as a Service) platform. These are meant to allow the user to focus on the client side of their application, by outsourcing the complex handling of data servers, storage, user authentication, and other features to their BaaS provider. They are very cost effective, significantly speed up development, and will already have a suite of refined and integrated tools you can immediately use in your own application. This includes features like storing users for login and signup, holding the data they submit, email, phone or other methods of authentication, push notifications, analytics, and many others.

While you technically could write and manage all backend code and data yourself (say if you didn't want anyone else to have access to your information) this is unrealistic for a new developer. I highly doubt your reading this guide if you already own your own server, so because we have to use another service provider's server, we may as well take advantage of the additional services they offer to expedite our development process.

Here is an overview of common BaaS platforms and a comparison between them:

Firebase - This is part of Google Cloud our recommended service and the one we will cover extensively through this guide. It offers both Realtime Database and Firestore which are each NoSQL databases, meaning they are highly flexible, allowing you to easily pivot on concept depending on what user data you store (as opposed to SQL, which is highly structured and hard to pivot, more here: <https://www.ibm.com/blog/sql-vs-nosql/>).

It is known for being very user friendly and easy for new developers to learn. It offers a free spark plan to test their products, and you can switch to the flexible Blaze plan, which is “pay-as-you-go”, so as you scale your platform, you're never paying for more services than you need. It supports all of the features we mentioned, which all come “out-of-the-box”, meaning there is no additional setup needed.

It also gives you access to Google Analytics, to see how users are accessing and interacting with our platform, crucial for a SEO scaling strategy. It is also built to be mainly used with JavaScript, which is conducive to the skills we have learned.

For these reasons, Firebase is highly regarded among startup founders, great for smaller projects but still scalable, and will be the best fit for our startup as well. More information here: <https://firebase.google.com/>

AWS - Amazon Web Services is the largest cloud service provider with the most global data centers. It may be overwhelming for new users as it is very complex due to a huge number of provided services. Many of the features, like databases and user authentication, do not come out of the box, and will require additional cloud knowledge and code to set up within your own project.

While it is compatible with many languages and very flexible for developers, this flexibility stretches beyond the scope of what is necessary for our product. Lastly, the pricing

model is harder to understand and may be difficult to predict anticipated costs. Overall, it will typically be better for experienced enterprises than startups. For more information, see here:
<https://aws.amazon.com/>

Azure - This is Microsoft's cloud computing platform and operates great within the Microsoft ecosystem. However, for new developers who wish to branch out, it poses many challenges similar to AWS. While it offers more services than Firebase, it will require additional knowledge and coding to implement these tools. The benefit of flexibility comes at the cost of complexity. This complexity is also reflected in their pricing model. For more information, see here:
<https://azure.microsoft.com/en-us/>

10.2 Node.js

Node.js is a runtime environment (defined earlier 7.5) that allows us to use our JavaScript skills to add functionality. While you may consider some of the previously mentioned alternatives for the backend, I highly recommend using Node.js, as it significantly decreases the learning curve, and is very compatible with our BaaS choice Firebase.

Firebase allows you to use npm or a CDN to configure Firebase in your project. A CDN can be included in script tags in your HTML, and is best for smaller projects and testing of simple (not complex) projects. The npm method is much more scalable, using JavaScript build tools, and will be more conducive to some of the complex features we are attempting to include, so I recommend that method.

For a quick overview of Node.js, I recommend the following video:

<https://www.youtube.com/watch?v=pU9Q6oiQNd0>. It covers what it is, how to use it, and how to install it straight from the terminal. It also covers how modules work (we will be importing these from firebase), how to use and install npm modules, and another method for creating a local server.

Another video you may find useful is the following:

<https://www.youtube.com/watch?v=ENrzD9HAZK4>. It explains NVM (node version manager), how Node itself actually works, and how you may incorporate it directly into a JavaScript file (using VS Code).

The following is a guide of the basics of Node.js and some common syntax you'll be using:

<https://www.linkedin.com/pulse/nodejs-cheat-sheet-jomas-daconceicao/>

Lastly, as you get more comfortable, you may want to explore the full API of Node.js:

<https://nodejs.org/docs/latest/api/>

10.3 Setting up Firebase

Now that you have a basic understanding of what Firebase does, and the purpose of Node.js, we can combine these skills to set up Firebase within our own project.

The first step involves creating a project. After that, you will need to install the Firebase CLI (command line interface) and Node.js (if you haven't already) through the terminal, and login. These steps are outlined here:

<https://www.youtube.com/watch?v=mmmaeHBCTOw>

Now that you are logged in, we can set up hosting so that our website is visible on the web. Firebase offers a free domain to do this. You'll want to navigate to your project's home directory in the terminal (the parent of your **public** folder). From here, you can initialize the directory with firebase, and begin testing locally using the *firebase serve* command. The steps are outlined here: https://www.youtube.com/watch?v=qbxj_36UkWs&t=404s

Note: This is an older interface of firebase terminal. In newer versions, when setting up hosting, you'll have the ability to also set up a CI/CD workflow with Github actions. This video covers the process and some ways to use it (7:01 - 12:53)

<https://www.youtube.com/watch?v=P0x0LmiknJc>.

In this tutorial, you won't need to set up CI/CD as we are working on a small scale project, likely individually. However, if you are working with a build tool (more below) or a team, you may find the automated process helpful. As opposed to just updating your repository locally, a workflow can allow you to update your code base, run a build tool, run tests automatically deploy or merge branches, and much more. When you have a large scale application with many moving parts, it is helpful to do this all at once as opposed to manually. Furthermore, it can increase the efficiency of a team with multiple developers, as you have a centralized, regularly updated codebase that will prevent overlap or interference. This can help locate bugs quicker and deploy working versions of your website faster.

If you realize your code base is becoming increasingly large, you may want to use a build tool. Instead of compiling everything during each deployment or update, it only recompiles the parts that have been changed. It automatically updates and tracks libraries or other dependencies you've used. It keeps compilation consistent, so that you won't have different bugs on different platforms or browsers. This can all be automated with CI/CD. For more information on why you might use a build tool, and some common ones to consider, see here:

<https://geekflare.com/node-js-bundler-and-build-tools/>

After initializing firebase, you will have a firebase source and json file in your parent directory. Your source file is just meant to specify what Firebase project you are linking to. Your json file tells your project what files to host and what to ignore. You can also set up additional

configuration and redirects if you want, however, leaving these files as their defaults will work fine for our project. For more information on editing your json file, see here:

<https://www.youtube.com/watch?v=Vt08efzophA>

This is an example of what the default files look like for CAC:

.firebaserc

```
{  
  "projects": {  
    "default": "cac-web-app-a3850"  
  }  
}
```

firebase.json

```
{  
  "hosting": {  
    "public": "Public",  
    "ignore": [  
      "firebase.json",  
      "**/.**",  
      "**/node_modules/**"  
    ],  
    "rewrites": [  
      {  
        "source": "**",  
        "destination": "/index.html"  
      }  
    ]  
  }  
}
```

```
"public": "Public",
```

This code says that we want to host all files within our Public folder

```
"destination": "/index.html"
```

This line says we want all links to our website to redirect to our homepage.

The last step involves actually deploying our website. This can be done simply by running *firebase deploy* in the terminal. You can also set the number of previous versions of your website you want Firebase to save, and rollback to those versions if you see necessary (perhaps if a new version isn't working correctly). For more information on how to do this, see here:

<https://www.youtube.com/watch?v=GzG1hqej7UQ>

10.4 Incorporating Firebase Tools

Now that we have Firebase setup, we can cover how to use it to add functionality to the frontend of our website and implement the features we specified earlier. Before starting, you may find it helpful to watch this video, which gives a high level overview of what Firebase Authentication and Storage is, and how they may be implemented into your project. You won't understand it all yet, but it will help in the future to see what we are working towards:

<https://www.youtube.com/watch?v=9kRgVxULbag>.

To cover feature implementation, we will be using more of Net Ninja's Youtube Tutorial. While seemingly unpopular, these videos cover these tools in a very detailed fashion oriented towards beginners, in even more depth than Firebase's own tutorials.

To start, you'll want to understand the logic behind how the Authentication works and interacts with the server, and how to initialize storage and authentication directly in firebase. These are covered here: <https://www.youtube.com/watch?v=mEF9WRwYDfY>

This video isn't completely necessary, but outlines what HTML elements will be interacting with the JavaScript, and where data will propagate. You may want to watch it if you're still uncomfortable with HTML div containers:

<https://www.youtube.com/watch?v=lkNvj7fUcRw>

In this video, he recommends adding your scripts to a separate file. I actually recommend against this, as most of your script will be page specific, and it's easier for beginners to see how it works, and debug, if it's directly in the same HTML file in script tags.

10.4.1 Configuration

After this, you need to configure your project so that it knows how to interact with firebase. This is done by importing Firebase libraries, and running a function to configure your project based on its identifying information. The following video outlines how to do this with the default generated index.html file: <https://www.youtube.com/watch?v=rOqvU46sdoU>

Contrary to the video, which configures firebase directly in index.html, I recommend moving this script alone to a separate file. Your website will be more efficient, such that you only configure firebase when it is called and needed. Furthermore, you can initialize commonly used libraries (that will be utilized on multiple webpages), and export them to be used multiple times, as opposed to having to import them on each individual webpage.

You can find your project's specific SDK (software development kit) setup by navigating to firebase > project settings > npm, and copy and pasting the given code into a separate file in your project named firebaseconfig.js. Here is an example, with CAC's sensitive information omitted.

```

// firebaseconfig.js
// Import the functions you need from the Firebase SDKs
import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.0.0.firebaseio.js";
import { getAnalytics } from
"https://www.gstatic.com/firebasejs/9.0.0/firebase-analytics.js";
import { getAuth } from "https://www.gstatic.com/firebasejs/9.0.0/firebase-auth.js";
import { getFirestore } from
"https://www.gstatic.com/firebasejs/9.0.0/firebase-firebase.js";

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "_____",
  authDomain: "cac-web-app-a3850.firebaseio.com",
  projectId: "cac-web-app-a3850",
  storageBucket: "cac-web-app-a3850.appspot.com",
  messagingSenderId: "_____",
  appId: "_____",
  measurementId: "_____"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
const auth = getAuth(app);
const db = getFirestore(app);

export { auth, db };

```

We import commonly used libraries, like the database and authorization. These can then be called from separate files as such:

```
import { auth } from './firebaseconfig.js';
```

10.4.2 Authentication Tools

Now, we want to add the ability to create new users. This should be done on the frontend Sign Up page you created. The following video covers the process, including how to use query selectors, to interact with a specific form in your HTML code. It also shows how to use event listeners, to respond when the client clicks submit, and prevent the default action of refreshing the page. The video covers how to do this using only email and password (the only current authentication method available on CAC), but you can also add additional sign in methods, such

as using a Google account. You will see new users immediately within Firebase. For the process, see here (which also covers the previously mentioned CSS modal style):

<https://www.youtube.com/watch?v=wkdCpktUfGg>

Here is how this sign in functionality was added to CAC's sign up page:

```
<script type="module">
    import { auth } from './firebaseconfig.js';
    import { createUserWithEmailAndPassword } from
'https://www.gstatic.com/firebasejs/9.0.0/firebase-auth.js';

    document.getElementById('signup-form').addEventListener('submit', async (e) =>
{
    e.preventDefault();
    const email = e.target.email.value;
    const password = e.target.password.value;

    try {
        const userCredential = await createUserWithEmailAndPassword(auth,
email, password);
        console.log('Account created:', userCredential.user);
        alert('Account created successfully! Redirecting to login page...');
        window.location.href = 'LogIn.html';
    } catch (error) {
        document.getElementById('signup-error').textContent = error.message;
    }
});
</script>
```

Note that this code will also throw an error if a sign in is invalid, such as if an account was already created with that email address. Your code may or may not automatically sign a user in when the account is created. For CAC, we don't sign them in, but direct them to the login page after successfully creating an account.

Adding login functionality is relatively easy, and the process is covered here:

<https://www.youtube.com/watch?v=JWeoQn6KB0o&t=405s>

In CAC, we added this functionality to our login page as such:

```
<script type="module">
    import { auth } from './firebaseconfig.js';
    import { signInWithEmailAndPassword } from
'https://www.gstatic.com/firebasejs/9.0.0/firebase-auth.js';
```

```

document.getElementById('login-form').addEventListener('submit', async (e) => {
    e.preventDefault();
    const email = e.target.email.value;
    const password = e.target.password.value;

    try {
        const userCredential = await signInWithEmailAndPassword(auth, email,
password);
        window.location.href = 'Profile.html'; // Redirect on successful login
    } catch (error) {
        document.getElementById('login-error').textContent = error.message; // Show login errors
    }
});

</script>

```

Our code will redirect them to their profile upon successful login, or throw an error message, such as if the password was wrong or if the email is not associated with an account. We can check if their account information is valid, by assigning variables to the email and password they type in, and checking it against our firebase servers with the following line:

```
const userCredential = await signInWithEmailAndPassword(auth, email, password);
```

After this, you'll also want to add a button that allows the user to sign out, or else they'd have to clear their browser cache to leave their profile. This is very simple, only a few lines of additional code using the same methods above, and is covered here:

https://www.youtube.com/watch?v=eS-yU_6aKEE

Here is how CAC added this to the profile page:

```

// Logout functionality
document.getElementById('logout-btn').addEventListener('click', () => {
    signOut(auth).then(() => {
        window.location.href = 'LogIn.html';
    }).catch((error) => {
        console.error('Sign Out Error', error);
    });
});

```

While not completely necessary, you may find a need to track when the user login state has been changed. This is done with the onAuthStateChanged method, and will be used later when we

cover Profile data. How to use and implement this tool is covered here:

<https://www.youtube.com/watch?v=5VxqV8FhlVg>

For example, we use it before checking if we should propagate Profile data on the profile page:

```
// Display user profile data
onAuthStateChanged(auth, user => {
  if (user) {
    const docRef = doc(db, "users", user.uid);
    getDoc(docRef).then(docSnap => {
      if (docSnap.exists()) {
        // omitted code
      } else {
        profileContainer.innerHTML = "<p>No profile information
found.</p>";
      }
    });
  } else {
    window.location.href = 'LogIn.html'; // Redirect to login
  }
});
```

10.4.3 Database Tools

In our closing walk through on Firebase functionality, we want to add the ability to add, retrieve and delete data a user uploads to our service. This functionality will be crucial to our marketplace, as it will the majority of content within our service.

To start, you may want to watch this video, which gives an overview of the logic on how data is stored in Firebase, using a NoSQL method:

https://www.youtube.com/watch?v=4d-gIPGzmK4&list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB&index=1&t=0s

Furthermore, you've likely already initialized your database above, but if you haven't, here is a walk through on how, and setting our security rules to test mode temporarily:

https://www.youtube.com/watch?v=UFLvSp4Mh9k&list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB&index=2

Next, the following video walks you through the process of creating data in firestore and retrieving it directly. These methods are not used directly in CAC, but it is still a good quick practice:

https://www.youtube.com/watch?v=kmTECF0JZyQ&list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB&index=3

We want to save data input into forms by the user. Previously, we used input data to check authentication status. Now, we want to store the data that is input (typed) by the client on

the form into our database, as opposed to reading and discarding it. We can do this with the add method, and save the value of each individual parameter, to create a new doc with the data:

https://www.youtube.com/watch?v=zpQle4SBRfg&list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB&index=4

Also, as opposed to creating an entirely new document, you may also just want to update individual fields, which you can learn how to do here:

https://www.youtube.com/watch?v=fekIsTaQ9Kw&list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB&index=9

In CAC, we used a slightly different method. Because all data is attached directly to the user (their profile data), when they edit it, we retrieve a document with the same name as their user ID, and use a query selector to edit each field individually, or create an empty string if the field was left blank. Here is how this was implemented in Edit Profile:

```
// Initialize Firebase Storage
const storage = getStorage();
onAuthStateChanged(auth, user => {
  if (user) {
    currentUser = user;
    const docRef = doc(db, "users", user.uid);
    getDoc(docRef).then(docSnap => {
      if (docSnap.exists()) {
        const userData = docSnap.data();
        // Update form fields with user data
        document.querySelector('[name="name"]').value = userData.name || '';
        document.querySelector('[name="sport"]').value = userData.sport || '';
        document.querySelector('[name="school"]').value = userData.school || '';
        document.querySelector('[name="classYear"]').value = userData.classYear || '';
        document.querySelector('[name="rate"]').value = userData.rate || '';
        document.querySelector('[name="bio"]').value = userData.bio || '';
        document.querySelector('[name="email"]').value = userData.email || '';
        document.querySelector('[name="phoneNumber"]').value = userData.phoneNumber || '';
      } else {
        console.log("No profile found!");
      }
    })
  }
})
```

```

        }
    });
} else {
    window.location.href = 'LogIn.html';
}
}) ;

```

Important to note is the use of docRef:

```
const docRef = doc(db, "users", user.uid);
```

This line will either retrieve an existing document, or create one if it doesn't exist.

CAC does not delete any user created data directly, but rather just replaces it as shown above.

However, if you need to delete data (an entire document), the following video should help:

https://www.youtube.com/watch?v=UZqXcoqC95E&list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB&index=5

If you are dealing with large amounts of data, you may find it necessary or helpful to add queries, such that only data meeting certain criteria will be visible to the user. This video covers how this can be done with the where method:

https://www.youtube.com/watch?v=mYyPbfpoZeM&list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB&index=6

While CAC doesn't query the data directly, it instead uses a search bar functionality, that allows the user to apply their own filter on visible data, depending on what they search. Profiles must contain the string the user types somewhere within it in order to be visible. This is implemented in our Find a Coach page:

```
async function loadProfiles(filter = "") {
    profilesContainer.innerHTML = ''; // Clear existing profiles
    const querySnapshot = await getDocs(collection(db, "users"));
    querySnapshot.forEach((doc) => {
        const userData = doc.data();
        const profileText = `${userData.name} ${userData.sport}
${userData.school}`.toLowerCase();
        if (!filter || profileText.includes(filter.toLowerCase())) {
            const profileBox = document.createElement('div');
            profileBox.classList.add('profile-box');
            profileBox.innerHTML =
                `![Profile Image](${userData.photoURL || './images/profileIcon.jpeg'})

${userData.name}



${userData.sport}

`;
            profilesContainer.appendChild(profileBox);
        }
    });
}
```

```

        <p>${userData.school}</p>
    `;

    profilesContainer.appendChild(profileBox);
    profileBox.addEventListener('click', () => openModal(userData));
}

);

}

loadProfiles();

document.getElementById('profile-search').addEventListener('input', (e) => {
    const searchTerm = e.target.value;
    loadProfiles(searchTerm); // Reload profiles with filter
});

```

Say if we wanted the data to appear in a certain order, like alphabetical under a certain field (exp: sport), we could add a collection index directly in Firebase:

https://www.youtube.com/watch?v=cb8H_hp10rc&list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB&index=7

If you want data to update in realtime, you'll need to use the snapshot method, covered here:

https://www.youtube.com/watch?v=RdnPAOA6qfw&list=PL4cUxeGkcC9itfjle0ji1xOZ2cjRGY_WB&index=8

We use this method in Find a Coach, in this line found from the code snippet above:

```

const querySnapshot = await getDocs(collection(db, "users"));
querySnapshot.forEach((doc) => {

```

This is meant so that when a user edits their own profile, or if they are on the platform and someone else creates a profile, it will automatically be shown.

Lastly, here is some other functionality we implemented you may find useful.

Modal CSS was covered above, but here is our method of propagating profile data in a modal environment on Find a Coach:

```

// Function to open modal and display profile details
function openModal(userData) {
    modalContent.innerHTML =
        `

    <span class="close">&times;</span>
    
    <h2>${userData.name}</h2>

```

```

<p><strong>Sport:</strong> ${userData.sport}</p>
<p><strong>School:</strong> ${userData.school}</p>
<p><strong>Class Year:</strong> ${userData.classYear}</p>
<p><strong>Rate:</strong> ${userData.rate}</p>
<p><strong>Email:</strong> ${userData.email}</p>
<p><strong>Phone Number:</strong> ${userData.phoneNumber}</p>
<p><strong>Bio:</strong> ${userData.bio}</p>
`;

profileModal.style.display = 'block';

// Close button inside the modal
modalContent.querySelector('.close').addEventListener('click', closeModal);
}

```

We retrieve all data fields directly from the user's data and display it into the modal environment. If the user has not uploaded a profile picture, we display the default profile icon as such:

```


---

## **11. Custom Hosting**

Through the steps above, your finished website should already be available through a custom firebase domain. However, it doesn't look great and you don't want customers to have to search for your website using that long domain. Instead, it's best that you have a custom website name, which you can get using the following steps below:

### **11.1 Acquiring a Domain**

There are plenty of options for choosing a service to buy and host your domain. Two of the most popular are [Squarespace](#) and [GoDaddy](#). Additionally, both of these platforms offer tools for website design and creation (which isn't useful for us since we are developing it ourselves). Personally, I use [DreamHost](#), which is primarily used for WordPress based websites, but can host other types as well. It really does not matter which platform you choose, so I recommend checking pricing and special offers at the time you're purchasing to find the best deal.

Check if your domain is available. You may have to alter the name slightly, or include dashes or other unique differentiating marks, if your name is already taken. For example, in our case, [www.CollegeAthleteConnect.com](#) was already taken, so I opted for [www.College-Athlete-Connect.com](#). You can hire a domain broker to acquire a specific domain if it's integral to your business plan, but this is usually very expensive and not necessary for an MVP.

For a specific business name, you can usually expect to pay in the range of \$10 - \$30 yearly (often times a lot less for the first year due to promotions). I recommend setting up yearly automatic payments to ensure you don't forget to pay and your domain name is taken.

Registering a domain is a relatively simple and self explanatory process within a service, however, if you need assistance, here is a tutorial specific to Dreamhost, see 5:00 - 7:05

[https://www.youtube.com/watch?v=Dr\\_H2fCPNLU](https://www.youtube.com/watch?v=Dr_H2fCPNLU). One advantage to using Dreamhost is that they will protect your information for free, such that people cannot see what email your domain was registered under. Other providers will usually charge extra for this service.

## 11.2 DNS Routing

Now that we have our domain registered through another service, even though they may offer hosting for that domain, don't buy it. All of your information is on Firebase, which is already offering free hosting. So, we need to redirect our domain to our Firebase project. We can do this by altering the DNS (Domain Name System) records through our domain provider.

First, you'll want to go into Firebase to find the corresponding DNS record you need to alter. To access this on Firebase, you'll just need to go into the Hosting tab and click add custom domain. See the first 1:25 of this video for assistance on how to do this:

<https://www.youtube.com/watch?v=IBMNvoJcy-k>

The above video also shows you how to change DNS records if you are using Google domains.

To alter our DNS records within Dreamhost, refer back to the video from 11.1, and watch 7:05 - 7:40 [https://www.youtube.com/watch?v=Dr\\_H2fCPNLU](https://www.youtube.com/watch?v=Dr_H2fCPNLU). Firebase should give you corresponding A and TXT records you will have to enter (if you choose a quick setup, there is no need for advance for our project). There is no need to alter the TTL (time to live) value, this can be kept as the default. After adding, refer back to Firebase. It usually takes a few minutes for your DNS records to propagate on the web, so wait for that to update, but after it's finished, you should be able to access your Firebase website directly through your domain.

## 11.3 Custom Mail Records

If you set up a Google workspace like we suggested, you will have the ability to set up a custom email address. This involves changing the MX records within your DNS settings the same way you altered A and TXT above. 3:53 - 9:38 of this video should walk you through how to do so: <https://www.youtube.com/watch?v=5BvxtZ4yQao>. Note that it may look slightly different depending on what domain provider you use, but it will still show you how to access your MX records.

If you are using Dreamhost, the process is slightly different. Instead of editing DNS, you'll want to click the Mail tab in the side panel, then Custom MX, and click Edit on the right side of your website. Here you'll see your MX records that you can edit.

Dreamhost does not have a separate field for the Priority number, which is necessary. To put it in, you'll use the following convention #\_record. The number, a space, and then the record

google gave you, all in the same field. Refer back to Google workspace, give it some time to update, and make sure Google successfully verified your MX record.

## **12. Conclusion**

### **12.1 Market Testing**

At this point, you should have a completed MVP that is published on the web and ready to begin market testing. Congratulations! This is a huge accomplishment.

As you release your product to the public, using whatever business strategy you employ, I recommend referring back to this guide to iterate your product. Chances are, there are a few bugs, which is expected. Use the resources within to help solve the bugs, or branch off as you see necessary.

This guide was meant only to aid you in the technical aspect of entrepreneurship. Now that you have a functioning product, you can use your business skills to conceive the best go-to-market plan to grow your company.

### **12.2 Final Message**

Thank you for reading this guide, and I hope it has assisted you in your journey to becoming a technically capable entrepreneur. Remember that technology is constantly improving, so you need to as well. But now, you have proved to yourself that technology is not a limitation, but an enabler, for bringing your ideas to life. Serve the public well with your new skill set.

Lastly, thank you for the previous work done by all of the educators I linked to throughout the guide. The internet is capable of being the best learning tools available, if you use it correctly, thanks to the countless contributions of generous minds. I hope this guide serves as one more contribution to the public knowledge on these subjects.