Web Development 101

Total Estimated Time: 60-90 hrs

Top level table of contents

Table of Contents

Jump to Mini-Projects (good for pairing) * HTML/CSS: Google Rebuild * Javascript: Sketch Pad * Ruby: Test First Ruby * Rails: Blogger Tutorial

Intro

Now that you know what web developers do, it's time to start thinking about how they actually do it. In this unit, we'll learn about how the Web works and start thinking about the basics of computer and web programming. You can consider this the "Prep Work" section of this curriculum but it will also serve as a reference for later. This unit will probably take between 60-90 hrs to complete.

Each of the following sections represents essential baseline knowledge. Even if you have no intention of becoming a web developer yourself, this material should help you gain a useful understanding of the moving parts involved in creating and serving content on the web. We will start by familiarizing you with the internet and your own computer. You will then learn the basics of front end technologies like HTML, CSS, and Javascript before stepping into the back end with a brief foray into Ruby and Ruby on Rails. We will finish by learning about databases and topics like git workflows and software in the cloud.

By the end of this unit, you should not only understand how the web works, but be able to identify and differentiate each of the technologies that you will be learning in order to build web applications of your own. You will be able to build a simple webpage, style it, and add minor elements of interactivity while working comfortably from the command line.

We will cover a very broad swathe of knowledge here and that's intentional... it's silly to go diving straight into server-side programming without having a context for what it is and why it's useful (and why you should learn it!). Each section will contain questions that you'd better be able to answer before moving on. For the first time, some of the sections below will ask you to do exercises and even a couple of mini projects to help build your understanding. While most of this material should be fine to tackle on your own, the projects in particular can be useful ones to find a partner and pair up on. Give it a shot!

Final note: Please let us know how long these sections take you so we can provide more accurate time estimates to future students.

How Does the Web Work?

Estimated Time: 1-2 hrs

You will need to understand:

- What is the internet?
- How is information broken down and sent?
- What are packets?
- What is a "client"?
- What is a "server"?
- What is HTTP and how does an HTTP request work?
- What are DNS servers?
- What is HTML and how is it used?
- What is CSS and how is it used?
- What's the difference between static and dynamic web pages?
- What happens when you click "search" on google.com?

Assignment:

1. For a basic and painless Internet overview, check out Don't Fear The Internet's Simple Primer (video).
2. Check out How Does the Internet Work? (video) for a basic explanation of how packets work.

3. Harvard's David Malan explains internet basics starting at 52:15 in this lecture from his popular Intro to Computer Programming course. Note the "developer tools" section of the browser that he talks about around minute 58… you'll use that a lot. We'll get heavily into HTML later in the curriculum.
4. He dives further into HTTP in a short 3-minute video (video).
5. Check out a bit more on developer tools from Don't Fear the Internet, since getting familiar with your browser's tools will help you out quite a bit as you start poking around the web pages you or others have made.
6. (Optional) If you want to learn a bit more about the amazingly useful developer tools for your browser, check out this 10-minute video that goes over them in some detail. A bit of knowledge here can save you a lot of time debugging later!
7. (Optional) For a more detailed (though a bit dated) text explanation of http, check out this link.

What's the best explanation of how the web works you've seen or read? Were any other resources you found helpful for answering the questions posed at the beginning of this section? Let us know!

Additional Resources:

- A packet's journey (video).
- An entertaining TED talk about how the Internet works.
- How the web works… a cartoon.
- Udacity has a quick explanation of web basics and HTML basics (these will require login/signup but it's free).
- A comprehensive list of web development resources lives at the Web Standards Curriculum.

Step Back… How Does Your Computer Work?

Estimated Time: 2-4 hrs

This isn't a course about rebuilding your hard drive but it is useful for you to understand the high level overview of what's going on to make your computer function. That's partly because you'll see the same patterns showing up when you're telling servers what to do and partly because you're going to have to talk to your computer in a way that it understands anyway so you'd better speak a little of its language.

Assignment:

1. Spare 15 minutes to learn how computers work.

The Command Line

Raise your hand if you're scared of the command line. Yes, we have this image of developers staring at a black screen with white or green text flashing across while they wildly enter incomprehensible commands and hack into the corporate mainframe (spilling soda and wiping cheetos off their keyboard no doubt as well).

That black screen (or window) is the command line, where you're able to enter commands that your computer will run for you. And while that image of a programmer is a bit overdone, the command line really is sort of like our base of operations, from which we'll launch other programs to do our "real developing" in. It has a syntax of its own which is different but not all that difficult to pick up. You'll be entering the same commands dozens of times anyway, so you'll get pretty good at it in a short period of time.

Assignment:

1. To gain a better understanding of your command line, take this Command Line Crash Course by Zed Shaw.

Want to help? Tell us how long that course took so we can let others know!

Additional Resources:

- See common commands on the Rails Beginner Cheat Sheet

Terms to Know

Estimated Time: 1-2 hrs

Assignment:

1. Take a look at this list of term definitions and make sure you have at least a competant understanding of each one (click on it to bring up the full definition). Don't skip this and assume you know it all – there will be at least a few that are worth nailing down. Flashcards help.

Web Programming Basics

Estimated Time: 40-60 hrs

Many of the things we're about to cover will be handled in much greater depth later on in the curriculum but it's important to get a basic level understanding of what they are and how they're used so you can still see the bigger picture while you're down in the weeds learning that stuff. Take the time to get to know each of the major building blocks of a web application, from the front end to the back end and into databases. And just because we're covering a lot of material doesn't mean we're not learning anything well – you'll be spending a good bit of time becoming familiar with the "101" level of each technology (each section may take 5-10 hrs to complete fully).

Front End Basics

Estimated Time: 20-30 hrs

The "front end languages" live in the browser. After you type in an address in the address bar at the top and hit Enter, your browser will receive at least an HTML file from the web server. That file will likely tell the browser to go and also ask for a CSS file and a Javascript file as well. Each of these languages performs a separate but very important function and they work harmoniously together to determine what the web page IS, how it LOOKS, and how it FUNCTIONS. And keep in mind that your browser handles figuring out how to make these files into a functioning webpage (not the server).

Assignment:

1. Get a high level overview of how all three languages (and jQuery) work together in a web page in this brief learn.jquery.com post on the Anatomy of a Web Page.
2. If you didn't the first time, check out the this 10-minute video on your browser's developer tools, which should make a bit more sense to you anyway now.

HTML and CSS 101 Estimated Time: 6-10 hrs

You Will Need To Understand (at least):

- Why do we separate HTML and CSS?
- What are classes and IDs (and how are they different)?
- What are elements?
- What are tags?
- What are attributes?
- What are forms?
- What is a div?
- What are selectors?
- What are properties?

- What are values?
- What is the Query String in a URL and what does it do?
- What is the difference between pixels and ems?
- How do CSS styles for a particular element get inherited? ie. how does an element get its “default” styles?
- What are two CSS attributes you can change to push a tag around on the page?
- What are the three different ways to include CSS in your project or use CSS to style a particular element?
- What is the default stylesheet?
- Why use a CSS reset file?

As you’ve hopefully learned before, HTML is the markup that contains all the actual stuff that a web page has. All the text on this page lives inside HTML tags that tell your browser how to order the content on the page. CSS tells the browser if you want to display any of those tags a particular way like, for instance, turning its background blue and pushing it a little to the left.

Assignment:

1. Do the Codecademy HTML/CSS track for a healthy baseline understanding of HTML and CSS. It can be helpful to take notes or make flashcards to keep track of the most commonly used elements.
2. Build the Android logo using just HTML and CSS by watching this video from at an amazing website called The Code Player, which is like a tutorial but it actually plays the code as the original developer created it. Use JSFiddle or your own text editor to create the logo alongside the tutorial.
    - Note: You will see border-radius, -moz-border-radius, and -webkit-border-radius all used identically. This is in order to overcome some of the differences between browsers. You probably only need to use border-radius to achieve the desired effect.
3. Learn about basic forms from this Treehouse video and use this w3 page as a reference.
4. Optional: Learn about your browser’s default stylesheet and CSS resets in this video. This is why there are some spaces that show up in your layout even if you haven’t specified CSS. Real developers almost always use a CSS reset to blow away the default stylesheet and let them work from scratch.

Want to help? Tell us how long that course took and how we can make it more complete so we can let others know!

HTML/CSS Mini-Project: Estimated Time: 2-4 hrs

Know (and try) these before starting * Two ways to move a div around on the page * How to stick a div onto the bottom or top of the page * How to see what color the background of something is on an existing webpage * How to grab the URL for an image from an existing webpage * How to center an element horizontally * Three ways you can include your CSS styles in a page * How to use classes and ids to target CSS at specific elements on the page * How to build a very basic form (even if it doesn't "go" anywhere)

For this mini-project, you'll deconstruct an existing web page and rebuild it. Don't worry if the links don't go anywhere and the search box doesn't do anything when you submit it. The goal is to start thinking about how elements get placed on the page and roughly how they get styled and aligned. For some of you, this may be the first time you've actually tried to "build" something in HTML (very exciting!).

Use the browser's developer tools (right clicking something on the page and clicking "inspect element" will get you there) will be your best friend. Build the page in a .html text file and open it in your browser to check it out (or try using jsfiddle.net).

- Easy Version: Build the Google.com homepage (the simple one with just a search box).

  1. Tips:
     - DONT BE A PERFECTIONIST! You're just trying to make it *look* like google.com, not actually function like it and it doesn't have to be spaced exactly the same way. Any dropdown menus or form submissions or hover-highlighting should be ignored.
     - USE GOOGLE! You'll probably run into roadblocks where you can't figure out how to do something so do what all good devs do... Google for how to do it!
  2. Start with just putting the main elements on the page (the logo image and search form), then get them placed horizontally. You can either download the Google logo or link directly to its URL on the web in your tag.
  3. Next do the navbar across the top, first building the content and then trying to position it. Check out how to build a horizontal CSS navbar if you're lost.
  4. Finally, put in the footer, which should be very similar to the top navbar.

  5. In general, do as much on your own as you can before relying on the developer tools (or viewing the page's source code) to help you along.

- Difficult Version: Build the Google.com search results page. You should be able to reuse much of your code from before if you started with that project. Again, don't worry about links to nowhere and forms that won't submit and hard coding the search results (which you'll have to do of course), just focus on placement and order of items on the page.

Note: All the classes and id's and names of elements that you inspect on Google's home page are nonsensical strings (like `<div class='srg'>`). This is because the code was **Minified** (see the Wikipedia entry here, which removes or shortens unnecessary characters and names to help the page load faster. The HTML (or Javascript or CSS) file will be smaller but the browser can still read it just fine.

Additional Resources

- Watch all the free HTML videos from Treehouse and take the quiz.
- Check out the quickie CSS introduction from the same people as well.
- If you want to see the art of CSS, check out the CSS Zen Garden, which collects submissions that use identical HTML and modify only the CSS but turn out wildly different (and beautiful).
- Read through Shay Howe's HTML&CSS Tutorial. Lesson 1 gives a solid overview and you can do the whole thing for a more in-depth understanding.

Javascript and jQuery 101 Estimated Time: 15-20 hrs

You Will Need To Understand:

- What is a scripting language?
- What is a variable?
- What is a string? An array? A boolean?
- What is the DOM?
- How can you interact with the DOM?
- What are events?

Javascript: As we saw before, Javascript is the in-browser code that gets run to make things on your webpage moveable and clickable, including the dropdown menus and hover effects you use every day. It's time to dive in a bit more and start getting your hands dirty with some code.

This is also the first time you'll get to do some actual programming (HTML/CSS just kind of sit there). The jQuery section will focus a bit more on applying your knowledge to real webpages.

Assignment:

1. Do Codecademy's Javascript Section 1("Getting Started with Programming"). If you want extra credit, do the full Javascript track... but we'll get to that in-depth later on in the curriculum.
    - Want to help? Tell us how long that course took so we can let others know!
2. Be sure to do the Codecademy Choose-Your-Own-Adventure Project once you've finished the learning section.
3. Go to jsfiddle.net and play around with their tool – it lets you type out some HTML, CSS, and Javascript and see it displayed for you right there. It's great for just testing things out or solving simple problems. Use the "Run" button at the top to run your Javascript.
4. Start thinking about how to use code to solve more logical problems. As much as web development is an expression of creativity, it's also based in problem solving (and job interviewers certainly know that) so you'll want to shake the rust off that part of your brain. Project Euler is a series of programming challenges that are best solved by using the power of computers (since many of them require you to perform simple mathematical operations on a very large scale). Some of the later problems require so much repetition and computing power that you would need to find a more elegant way of solving it than the immediately obvious "brute force" solution. We'll just do a couple of simpler ones here. Solve these problems (try using jsfiddle if you aren't comfortable working on your own):
    1. Problem 1: Multiples of 3 and 5
    2. Problem 2: Even Fibonacci Numbers
    3. Problem 3: Largest Prime Factor
    4. No one said you could turn off your brain!

Additional Resources: * More videos about Javascript Functions from wickedlysmart.com * Reading: The first several sections of the Javascript 101 tutorial on learn.jquery.com. * Interactive: Do the additional sections in the Codeacademy Javascript Track.

jQuery: What about jQuery? It's a library of commonly used javascript widgets and functions that has more or less taken the internet by storm. It's written in javascript and it means that you don't have to go through the pain of building a popup modal dialog box or a dropdown menu, for example, the long way. It also gives you the incredibly easy ability to select elements on the webpage ("DOM elements") so you can start modifying their properties, whether that's hiding them, moving them, changing their contents... it's all in your hands!

jQuery will let you take your javascript knowledge and start really diving into your webpages and messing with the elements.

Assignment:

1. Learn about jQuery by doing Codecademy's jQuery Section 1("Introducing jQuery") and Code School's try jQuery. The remaining Codecademy sections are, again, extra credit and will be covered later in the curriculum.

Want to help? Tell us how long that course took (whether just the section 1 or the whole thing) so we can let others know!

TODO: Javascript/jQuery walkthrough video

Additional Resources

- Suggest some!

JS/jQuery Mini-Project: Estimated Time: 3-6 hrs

Sketch Pad * Create a web page (or use JSFiddle) with at least a 32x32 grid of square divs. Set up a hover effect so it changes the color of the square when your mouse passes over it, leaving a trail through your grid like a pen would. Have a button at the bottom to reset the grid to blank white. It may be easier to make borders between grid elements while you're figuring out how to place them.
Then try cranking up the granularity of the grid (say, a 128x128 grid of the same total size). Hopefully you've utilized a loop to create your grid and this should be a simple matter of changing a few inputs. * If you've got a cool solution, send it (or at least a screenshot) to curriculum@theodinproject.com so we can show it as an example!

Back End Basics

Total Estimated Time: 20-30 hrs

The three languages of the front end are fairly standardized – HTML for presentation, CSS for markup, and Javascript for scripting. The back end is a different story… you can run pretty much anything you want to on your server since it doesn't rely on you user's browser understanding what's going on. All the browser cares about is whether you've sent it properly formatted HTML, CSS and Javascript files (and other assets like images). That's led to a whole lot of different choices for back-end languages. As long as it can take in an HTTP request and spit out some HTML, you can probably put it on a server somehow.

That said, some languages are more popular and practical than others. If you're running your own server, you have a ton of flexibility but plenty of headaches. If you're deploying to the cloud (which we will be doing later), you may be restricted to those languages which your cloud provider has installed on their platform… it doesn't do you much good if the servers you're "borrowing" from them can't understand your code! Some of the most popular are PHP, ASP.NET, Ruby, Python and Java (not to be confused with Javascript). And just like I can say "which way to the nearest pub?" in Swedish, French, Italian, English and Bad English, all of those languages can perform almost exactly the same functions.

One difference between the work you did on the front end and what you'll be doing on the back end is that you'll need to make sure you have some things installed prior to actually building these projects in the back end. Your browser automatically knows how to run HTML, CSS, and Javascript for you but your computer probably doesn't have Rails installed.

Assignment:

1. Go to the Installations Unit and get everything installed if you haven't already. It can sometimes be a frustrating process but you'll need to get it done eventually so don't wait. That includes setting up git and your Github account, even if you're not quite sure yet what they are.

Ruby 101 Estimated Time: 20-30 hrs

You Will Need to Understand:

- What is an "interpreted" language?
- What is IRB?
- What are Objects?
- What are Methods?
- What are Classes?
- What are Blocks?
- What is an Array?
- What is an Iterator?
- What are hashes?
- What is a lambda?
- What is a proc?
- What is a library?
- What is a gem?

Our back end focus will be on Ruby, the language designed for programmer happiness. What takes dozens of lines of code in Java or a hundred in C could take just a couple in Ruby because it prepackages lots of sneaky functions into easy-to-use convenience methods. It has occasionally been criticized for not running as fast as some of the other languages, but that's not so much an issue when you can just load more servers/instances running your web application. Being able to write code faster and with less headache allow you to iterate frequently when building a website and so the end product is more likely to suit the client or the user's needs.

Ruby is pretty darn close to Python. In some ways, they sort of resemble romance languages – once you've learned one, it's not terribly hard to pick up another because they tend to follow many of the same conventions, just using different "words". Python tends to be taught more in colleges and is used a fair bit for more data-intensive applications. But Ruby has a secret weapon that makes it the love of fast-iterating website producers – the framework Ruby on Rails (which we'll cover in the next section on Frameworks).

With either of the languages, there are a couple of things that aren't immediately intuitive but become very useful when you understand them. We'll do a healthy introduction to Ruby here and then, in the unit on Ruby, you'll get to understand it like the back of your hand.

A final note – you'll be learning a bunch of new terminology and concepts here but don't think they're only applicable to Ruby. Most of it (like methods, classes, objects etc.) will pop up again in Javascript and pretty much any other language you pick up.

Assignment:

1. Go to tryruby.org and do the quick exercises there to get your feet wet.
2. Read through the Ruby in 100 Minutes project from Jumpstart Labs. If you can't get IRB running, check out the Installation Unit, which you should have done already.
3. Dive in a little deeper with Chris Pine's Learn to Program. Try to do the exercises at the end of each chapter. Take a crack at chapter 10, but don't feel disheartened if it still doesn't click for you.
4. Finally, conquer the Ruby Monk's Introduction to Ruby. If you're shaky on Hashes, Blocks, Procs, Lambdas, Modules, and I/O... you're not the first and won't be the last so have no fear. We'll dive deeply into those in the coming units.

Bonus: Redo the same Project Euler problems that you previously did in Javascript but using Ruby instead (try using IRB or a .rb file that you run from the command line): 1. Problem 1: Multiples of 3 and 5 2. Problem 2: Even Fibonacci Numbers 3. Problem 3: Largest Prime Factor

TODO: Ruby walkthrough video

Ruby Mini-Project: Estimated Time: 4-6 hrs

It's time for some Test-First Ruby! That means you'll be downloading a bunch of test files and your job will be to make them pass. It's a great way to start writing some real Ruby scripts and learn some testing at the same time. First, you'll probably want to skip ahead and read about testing in the section below called "Testing 101". There's a homework assignment there but it shouldn't take long. Or just wing it…

This exercise will involve a lot of figuring things out. There aren't very detailed instructions for what to do, just the batch of tests that you need to make pass. You need to look at those tests, figure out what they want, and write the code to pass them. It can be tricky to get the hang of at first but once you start making those tests go green it starts feeling pretty cool.

1. Go to testfirst.org's Learn Ruby section and follow the installation instructions. Basically, you'll be either cloning their Github repository or downloading a zipped file containing all the test files (there are 15). By opening the main index.html file in a browser, you'll have instructions waiting for you that describe what's going on, how to run the tests, and where to put your code so the tests can see it. The exercises start easy but some of them can be pretty challenging, so good luck!
2. If you absolutely must use an in-browser environment (it would be better for you to do this on your own machine), one is available specifically for these exercises at http://testfirst.org/live.
3. Do the following exercises. You'll know you're done when all the tests pass!

    1. 00_hello
    2. 01_temperature
    3. 02_calculator
    4. 03_simon_says
    5. 04_pig_latin
    6. 08_book_titles
    7. 09_timer

How was this? It should be difficult but do-able. How long did it take? Please shoot us a quick email to let us know.

Additional Resources

- The Railsbridge Ruby curriculum.
- Textbook: Peter Cooper's Beginning Ruby chapters 1-8 will cover the material in greater depth than you really need to just yet but may help you shore up some of the concepts.
- See the Rails Beginner Cheat Sheet for a condensed list of common commands and concepts.

Databases

Estimated Time: ??? hrs

You should know:

- What is a database?
- What are relational databases?
- How are relational databases different from XML?
- What is SQL?
- What does CRUD stand for?
- Why is CRUD so important for web apps?

We've talked about the client-side and the server-side but how do we keep ahold of all our user's data? Who remembers that your login password is CatLover1985 so you can sign into the website? The bottom layer of any web application is the database and it handles all the remembering for you (we'll cover caching much later). It can be relatively simple, like an excel spreadsheet, or incredibly complex and split into many giant pieces like Facebook's.

Databases are one of those aspects of web programming that are kind of hidden in the back and so people treat them with a sort of suspicion and awe. That's nonsense and you should get over it – your database and you are going to become very good friends (or frenemies). By the end of this curriculum, you're going to understand what's going on with your databases and be able to interact with them like a pro (and probably better than some people you'll work with).

Compared to a normal language, SQL (Structured Query Language), which is used to query databases, is a very simple syntax… there are only a small handful of verbs to learn. What trips people up is that you need to be able to visualize in your head what it's going to be doing. We'll spend a fair bit of time on SQL and databases because they're so fundamental, but for now we'll just cover enough to get you familiar with what's going on.

Assignment:

1. Check out a pretty good plain-english explanation of SQL, written by Zed Shaw, at Learn SQL The Hard Way.
2. Sign up for Coursera's Intro to Databases course and watch the lectures on Relational Databases and SQL (~3 hrs). You can save the sections in between (relational algebra, XML, JSON) for extra credit (things get a bit technical there).
3. Do the Automated Assignments for the SQL section to get a jump on your understanding for the SQL unit we'll be doing later. Try working with a pen and paper next to you to draw pictures of what's going on. Again, you can save the additional sections for extra credit.

Additional Resources: Do you know of a good plain-english explanation of databases? Email us or fork, add it here, then submit a pull request.

Frameworks

Estimated Time: 10-15 hrs

You'll Need to Know:

- What is a framework?
- What's the difference between a programming language and a framework?
- What languages have frameworks?
- What is open source software?
- What is MVC?

If you're programming with Ruby or any other language, you pretty much start with a blank file and go from there. Programmers, the best of whom are pretty lazy folk, got tired of having to write the same code over and over and over again just to cover the basic tasks that they wanted their applications to perform. So they batched that recycled code together and called it a framework.

Another benefit of using frameworks is organization. They tend to force you to organize your files and code in a way that keeps it highly modular and really clean. When you start a new Rails application, you're given dozens of folders already organized in a heirarchy which makes sense and follows good Model-View-Controller (MVC) separation principles. It's not quite "color-by-numbers" for code but it certainly keeps things ordered.

There are often several different popular frameworks for a given language. They can have exciting names like Ember, Meteor, Django, Rails, Grok, etc. Wikipedia has a comprehensive comparison of frameworks that should give you an appreciation for the number of them.

A final thing to note is about licensing – frameworks are typically open-source and their license allows you to use them, modify them, make money off them, sell products with them etc. all without owing any fees to their original creators. You may not think too much about that as you blithely code away using other people's frameworks, but it's a very important distinction between open-source frameworks and commercially produced/sold software. We'll get more into Open Source Software (OSS) in later units.

Ruby on Rails 101 Estimated Time: 8-12 hrs

You should understand:

- What is Rails?
- What are the seven components of Rails?
- How is a GET request different from a POST request?
- What is REST?
- What is the command to create a new Rails app from the command line?
- What are gems?
- What is the purpose of the gemfile?
- What is a view?
- What is a controller?
- What is a model?

You're probably here because you want to learn the Ruby on Rails framework but you may not be entirely sure what it is. Well, Rails is just a bunch of ruby code written to handle the parts of a web application that you don't often want to think about. Rails uses, as you'll often hear, "convention over configuration". That means that Rails has made a lot of decisions for you about how things should be structured and how the code should run. You can change them, but it's best if you just go with the flow and work within their rules (especially as a noob).

Think of it like buying a suit – you probably don't care where the thread was sourced, which hands pulled the loom, which companies shipped the fabric, how the lining is mated to the jacket, what sorts of buttons are on there… you trust the tailor to have figured all that out and you just want to be able to buy the nice looking grey one in size 42 regular. Rails is your Ruby tailor.

As you saw in the introduction: >Why Rails? Why not. There are dozens of possible technologies out there to choose from and, frankly, they do pretty much the same things. Rails is attractive because it's a relatively straightforward and very well documented framework that's used by tons of great startups and tech companies today and it has a very strong community of developers and students who support it. It lets you put up a functioning website in hrs not days or weeks. The "in" tech will probably be something completely different in a few years, as it always is, but Rails presents a great platform on which to build the skills you need to carry you to that next phase.

Because Rails has made a lot of decisions for you, you can work incredibly fast. You can have a website up on the internet (though it won't look like much) within a couple minutes. The very first time you generate a new project, everything is in place so you just have to fire up your local server (by typing simply `$ rails s`) and you'll already see the Rails welcome page up there. Then it's just a matter of plugging in all the pieces you actually need to make your rich web application run. It also means that you can start immediately making small changes and seeing how they affect your application where before you would have had to build a ton of infrastructure and write lots of code before seeing a single thing change live. Rails makes your life a whole lot easier!

Rails also firmly organizes your code using an MVC pattern which you will come to know and love.

The best way to understand Rails is to use it, so we'll spend a bit of time on some videos and reading but you'll mostly be building your own Rails sample app. You may have no idea what you're doing, and that's okay, but at least you should begin to understand what you DON'T know and what you'll want to pay attention to going forward. A good tactic is to write down all the things that confuse you and either go looking for them on your own or keep them in mind for later when we do our deep dive into Rails.

Assignment:

1. Watch this basic overview of Rails from Michael Hartl. It will walk through the creation of a very basic web application.
2. Read Daniel Kehoe's excellent What is Ruby on Rails? introduction to get a good grasp on what we're working with.
3. Start doing Rails right away by diving into the Rails for Zombies course, which lets you begin programming Rails right in your browser! It goes pretty quickly and you'll probably need to rewatch the videos but stick with it.
4. Read the Ruby on Rails Guides: Getting Started and try to follow along with the application they build (you don't need to build it, but try to read it through. You'll be building soon enough). By the end, your head will probably be spinning a bit but don't worry, that's normal. You'll understand this stuff no problem by the time we do our deep dive during the Rails unit later in the curriculum. The Ruby on Rails Guides provide some of the best documentation for the Rails language out there, so it's good to start getting familiar with how they look.

Rails Mini Project: Estimated Time: 3-5 hrs

- Do the Jumpstart Labs Blogger Tutorial.

Extra Credit: Michael Hartl guides you through creating a sample app using scaffolding as the first stage of his Rails tutorial. Just do chapter 2 – you'll be doing the rest of the tutorial anyway later on during the Rails unit. If you aren't comfortable with git, either do the section below on git (under the "Additional Topics" heading) or skip any of the `$ git ...` commands for now, they're non-essential (it's sort of like saving your files on Dropbox).

Additional Resources: An older and slightly more technical 1.5 hour video introduction to Rails from Armando Fox of UC Berkeley.

Backbone.js 101 Estimated Time: 1 hr

You thought only server-side languages could utilize frameworks? Think again. More and more heavy lifting is being done in the browser these days and that means lots of repetitive and disorganized code to tidy up (and Javascript can get awfully disorganized). While there are many popular Javascript framworks, we'll focus on Backbone for similar reasons to why we're focusing on Rails… it's probably the most straightforward and best documented one out there. That doesn't mean it's the shiniest or the newest, but if you've mastered your fundamentals in Backbone you'll be more than capable of picking up a slightly more esoteric framework. And there are a whole lot of websites out there that need to be built and maintained in Backbone (ahem, jobs).

This is probably the most "advanced" material we'll be taking a look at during this Web Development 101 unit since it relies most heavily on understanding a lot of things beforehand (like Javascript, MVC, frameworks, JQuery, APIs…) before you're really able to appreciate the need for it. Because of this, we won't be doing any exercises or anything like that, we're just trying to get you exposed to what Backbone looks like so you can start wondering what you're going to need to use it for and how later on.

Assignment:

1. Read through the nettuts backbone explanation. It should be somewhat confusing but you should also notice some structural similarities to Rails, even though this is on the front end. Again, don't worry if you're confused by what you're reading. The point isn't to absorb everything, it's just to kind of get a feel for what's waiting. We won't cover Backbone until fairly late in the course when you're a Jedi Javascript Master.
2. Also check out the CodeBeerStartups Backbone guide chapters 1 and 2 for some slightly more plain-english explanations. Again, don't panic… you'll learn this later.

Additional Resources

- The Backbone Documentation

Tying it All Together

Estimated Time: 1 hr

You've got an idea of what the pieces look like, now it's time to see how they all dance together in a typical Ruby on Rails application. Read through The Anatomy of a Web Application (you can listen to audio of it by clicking the link on the left side) and pay particular attention to what's happening on the client vs the server side. It gets a bit more into the guts of what Rails is doing, and this will be a valuable first step towards understanding what you'll be building later on.

Additional Important Topics:

Git 101 Estimated Time: ??? hrs

We mentioned git briefly in the previous unit (Becoming a Web Developer) but this time you'll actually start using it.

If you recall, git is the version control system used by developers. It means that you can revert to a previous (working) version of the application if you really screwed something up and it means that you have a standardized way of managing a project with contributions from multiple developers. It's even useful if it's just you and you're working on a project on your own with a few half-baked ideas that you're trying to code at the same time but want to keep separated until they've matured sufficiently.

It sort of feels like if you were typing a text document and, every time you saved it, you entered a summary message like "just finished paragraph on how git works". Then, if you realize at the end of the day that all your changes are awful and ruin the flow of the document, you could go back to the last save of yesterday and bring the document back to the way it was.

But wait, you say, Why not just erase the offending paragraphs and move on? Here's where the web application stops resembling a high school essay. Changes you make to your web application will be scattered in a dozen different files and will likely involve changing existing code at least as much as it involves actually producing new (and easily deleted) code, so having that ability to just reset the clock to a particular point in the past saves you from having to remember exactly what was changed and what it looked like beforehand. The power of git goes well beyond this simple example, but it should help you start to see why people use it.

Assignment:

1. Read through Github's Introduction to Git materials and watch the video there.
2. Do the quick Try Git exercises as well.

Testing 101 Estimated Time: ??? hrs

Like git, testing is one of those things that people often don't think about when they decide to learn about web development but it's critically important to the production of professional quality applications and it will save you tons of time and headache in the end.

Let's say you're building a simple website and you've got a couple of pages linked together with a simple navigation bar at the top. You make some changes to the code and want to know if the website still works. So you open up your local version of the webpage and click through each of the buttons on the navigation bar to make sure they each still lead to the right location. Not too difficult, is it? It doesn't take too long, and it doesn't seem like such a bad way of doing things.

But now imagine that you've got a dozen pages with a login system and content that's meant to look different depending on which type of user you are logged in as. You could come up with a checklist of all the buttons you'd have to manually click on and all the times you'd have to login as a different user but think about how many steps it would take before you were satisfied that your changes didn't accidentally blow up some obscure but necessary function of your website? Situations like that should yell "automate me!" in your head, and that's exactly what testing does.

On the back end, you will learn RSpec, a language that is written in Ruby and one which will help you execute a broad and flexible script of tests to make sure your application is still working the way that it should. RSpec's syntax even reads sort of like English, though it still takes some getting used to. RSpec lets you test specific areas of your application like pulling from and saving to the database or that your helper methods work as expected. With the help of a few useful gems, you can also test the broad-scale interactions the user will have when traveling from one page to another.

On the front end, we'll use a language called Jasmine to do a similar sort of thing.

Everyone does testing a little differently. Some teams still rely heavily on a Quality Assurance (QA) department with people manually executing checklists like we described in the example above. Some people use an approach called Test Driven Development (TDD) in which they write the (failing) test first and only then do they actually write the code necessary to make it pass, and thus very deliberately build the application out with 100% test coverage. Others prefer to keep their test suites fairly light and will only write tests for the major interactions on their pages and any bugs that they have to fix along the way (to make sure they don't come back). Regardless, testing is highly important and you'll be quite familiar with it by the end of this curriculum.

You'll be interfacing with tests before actually writing your own… how else do you think we process code submissions? Many exercises and projects will start with a batch of tests and have you write the code necessary to make them pass.

Assignment:

1. Do the Introductory RSpec level from Code School.

The Cloud, Hosting, and Software as a Service (SAAS) Estimated Time: ??? hrs

You've almost certainly heard of the Cloud before and have maybe also heard of SAAS. The Cloud is really just a way of saying "stored out on the Internet somewhere", which really means "stored on someone else's systems that they let us access using the Internet". With more and more of the world in posession of 24/7 internet access, that's not necessarily a bad thing. For software, it has been a great thing.

Check out this explanation of cloud computing from CNBC.

Not too long ago, you had to buy a machine, set it up to run your back end code, and plug it into the wall to get it onto the internet. These days, not only can you host your personal songs and files and emails in the Cloud, but you can actually run your website from servers hosted by someone else (and maybe in another country too). Companies (like Heroku, which we will use later) make it so that all you have to do is push some code their way and they'll take care of firing up and maintaining the servers necessary to get that application "live" and online. They will also guarantee a certain minimum level of uptime (usually well north of 99%) so you don't have to worry about whether your servers have failed. It means that you as a developer can focus more on building cool applications and less on the nuts and bolts of getting them out there. This curriculum assumes that you'll be taking advantage of existing cloud hosting, so we won't spend much time teaching you about how to set up your own servers.

Where before you had to write a piece of software that would work on a particular type of computer (e.g. Windows machines running Windows Vista) and then get that user to install the software, now many of these applications are run completely using websites. Think about Google Spreadsheets or Evernote or TurboTax... all of these can be run completely online (though they often have accompanying downloadable applications to help you out).

The ability to create this "Software as a Service (SAAS)" gives you tons of great flexibility as a developer – it means you no longer have to think about the dozens of different types of operating systems and versions that the users could be using. You just make sure your application can be viewed properly by people using a few different web browsers. Even better, when you inevitably want to make large-scale changes or just fix some bugs, you no longer have to convince your users to go through the painful upgrade process. Just push the changes to your sever, and Presto!, your application is updated and good to go. If there's a problem with the new Ford car model, they need to issue a recall and have everyone go to a mechanic to fix it. You just push some code to fix the bug and sleep happy.

Check out this article on the difference between SAAS and Cloud Computing if you're confused.

There's also another interesting perspective on SAAS from a few years ago.

Additional Readings

- An Intro to Cloud Computing by Craig Dickson
- Cloud computing for dummies

Security, SSL, and Best Practices Estimated Time: ??? hrs

TODO: SSL and Security thought questions

It's important to start thinking about how the open and bountiful world of the Internet handles issues of security and secure connections. Security is something that beginning developers don't spend a lot of time thinking about because they've got a lot on their minds but it will occupy more of your time as you start putting real applications onto the internet. It's something that you need to be familiar with because sometimes simple but incorrect choices can leave your users' data exposed or your application vulnerable to attack. Some simple best-practices and best technologies go a long way towards alleviating those issues.

There are a couple of basic areas where security is particularly important – authenticating users, creating secure connections and securing your databases. Luckily, the tools we'll be learning have already figured out good solutions to most of these problems.

Smashing Magazine showed some common security mistakes in an article from 2010. It references PHP code but the vulnerabilities are language agnostic.

To understand the basics of HTTPS and SSL, which help secure transactions on the web (like payments), check out the HTTPs Wiki and this article. Here's a basic explanation:

You want to pass a note from you all the way across the room to Suzy. Normally, you just pass the note and say "get it to suzy" and the kids in the room will keep pushing it towards her until she gets it. The problem is, the teacher or anyone who gets the note can just open it up and read it.

SSL is a type of certificate used to make sure the contents of a packet (note) don't get read. It's like putting your note in a lockbox and you've given Suzy the key ahead of time. She's the only one who can see what's in the box, because she has the key (the SSL certificate). HTTPS is an altered version of the HTTP protocol which makes sure whoever tries to open the box has the key. If anyone tries to read the note and they don't have the key, all they'll see is garbled (encrypted) data, which will most likely just look like random characters. it's like they took the box and just tried smashing it on the floor, but it ripped the note apart in the process.

FTP

Estimated Time: .5 hrs

FTP stands for File Transfer Protocol and is basically a way of transferring files to and from servers. Depending on your workflow, you may or may not use it directly but you should know what it is regardless. It is explained well in this FTP For Beginners post from Webmonkey.

How Are Websites Built?

TODO: In Becoming section instead?

Macro Level: From Client to Developer and Back

TODO: client team handoff, scrum, cards/stories, gogo, circle back, scrum.... debrief.

Micro Level: What You Do as the Developer

TODO: tech used, who gets spoken to, where debugs are usually needed, when tests are written, how deployment goes, interactions with other team members.

Principles of Good Programming

Estimated Time: 1 hrs

There are a handful of generally accepted principles of good programming. You'll hear them used by name more than a few times in the coming months but it's good to get familiar with them up front. Christopher Diggins has compiled a healthy list of them in this blog post that you should check out.

Other "Dumb" (or frequently asked) Questions Answered

Do you still have any beginner questions left over? Email us or fork, add them here, then submit a pull request.

Finish

Now you've gotten your feet wet and I hope you're getting curious to dig deeper. You should be fairly comfortable typing commands into your computer's command line and you should understand what happens when you enter a web address into your browser and it returns a functioning page. You should be able to create a basic web page and make it do some basic things with Javascript. You should also be able to write a simple Ruby script, test it, and describe what Rails and Backbone actually are. Finally, you should have a clear understanding of how all this fits together in your future as a builder of web applications.

With what you know already, you could start putting together some basic websites or further your knowledge on your own. But we're here to help you get a lot further than that – to the point where you can confidently build a full-featured and scalable web application, whether on your own or working as a developer in a top tier tech company. Buckle up, because we're heading straight into Ruby and we won't come out until you can confidently build a Twitter spambot and a chess game (among many other things). The journey has just begun!

Check out the next section on Ruby

Top level table of contents

Additional Resources

If there's anything we should have covered better (or at all), Email us or fork, add it, then submit a pull request.