SIMPLE, CHEAP, UNCRASHABLE

SHOPPING

A SITE DEVELOPMENT TUTORIAL

LEE DONAHOE

Simple, Cheap, Uncrashable Shopping

A Site Development Tutorial

Lee Donahoe

Contents

Why I wrote this

I am a designer and a front-end developer, and over time I've noticed an increasing number of online requests from fellow designers (or people new to web app development) for advice on what they should language or framework they should use to build a shopping site. Usually framed as a question like, "I hate my job and have this great idea to sell widgets, but I've only designed sites, never built one. Should I use Rails and build my own from scratch, or use something like Shopify, Magento, or BigCartel?"

Reading through the discussion that inevitably follows you quickly see that all of those options have pluses and minuses, but most end up giving you a system that requires a store owner to either pay create a complex custom application that uses a tradition application server, or pay extra monthly (or even by the item) for the privilege of using software that someone else has created.

In all cases though, if you have a product that goes viral you'll be at risk of having your server crash or paying a whole lot of money just to keep things up and processing sales – custom applications aren't easy for beginners to scale, and third party shopping apps eat into your margin by scaling up costs for resources as sales increase.

Granted they are providing you with some services like on call technical support (sometimes for an additional fee), and add-ons like templates, but ultimately they are banking on the fact that most business owners don't know how to create their own sites. Even if a hopeful entrepreneur did know how to build the site it is unlikely that they would be better at scaling them up to handle large amounts of traffic than the services offered by commercial shopping applications.

I want to change that.

What's the backstory?

The initial idea for this book come out of an application and technology stack that my company created for an online men's clothing company.

We'd been kicking around the basic concept for the project for about six months and hadn't had the chance yet to build it out for a client. The idea was to use a slightly unconventional combination of flat files (simple HTML, CSS, and Javascript with no application server), cached and served from redundant content delivery networks (CDNs), a cloud database as a service for managing products and users, and a server running our delayed processing to handle order completion and mailing users. If we did our job right, nothing in the stack would be traffic sensitive.

Our first real test came when the Wall Street Journal came out with an article (in the print and online versions) featuring one of their products, saying it had great style for a fraction of the price of designer version and to boot it was made entirely in the USA. Sales went crazy: over a 24 hour hour period they did $100,000 in sales and sold out their entire stock of inventory.

One thing didn't go crazy that day though: the application handled sales from hundreds of simultaneous users as if it was still just a couple developers putting through test orders before public launch. The technology didn't even have a hiccup in load times, even with 200,000 page-views in the rear-view mirror.

"So what," you might say, "Amazon does $100,000 in sales in the time it takes me to sneeze, and lots of small and middle sized businesses manage sales of similar volume." I'd be very willing to bet though that they aren't putting through that amount of sales on an application that costs a tiny amount of money, won't require additional upgrades or servers to handle future traffic, and needs precisely zero ramping up of resources to prepare for a major publicity event...

Github Pages, Stripe, and Jekyll

So I started thinking, "how can we change this model, and scale it back to cost as little as possible while still retaining the basic strengths of the bigger system?"

I figured that there is a significant amount of demand out there for a super cheap system that doesn't have the full feature set (like point-of-sale integration, inventory management, etc), but can still sell products... Something that would be ideal for mom and pop shops, or to quickly validate products without needing a full blown content management system to handle adding products, or complicated user management systems – just plain old sales.

It turns out that you no longer need to spend a ton of money on hosting when a $7 per month GitHub account lets you create a fully customizable site with their GitHub Pages feature. For your site to drop due to popularity you'd need to send it enough traffic to take down Amazon Web Services – not too likely.

You no longer need complex business logic running on a separate application server to handle completing payments. Stripe's API and a bit of Javascript magic take all the pain out of setting up payment processing, and it even handles mailing customers their sales confirmations.

You no longer need to use a complex CMS to handle filling out your site with products and copy; especially if you are creating a store just to handle a limited number of products. With just a little bit of setup ahead of time that involves a little code-work (no scary databases, I promise), you can create an incredibly easy to change web application using the Jekyll templating framework.

You don't even need to fiddle with buying and setting up SSL certificates anymore in order to serve a secure site. With a quick DNS configuration and a CloudFlare account you can have an automatically edge-cached and SSL protected shopping site.

The best part of all of this is that other than the fixed $27 a month you pay for a GitHub account and Cloudflare (actually drops to $12 a month for a second site since Cloudflare only charges you $5 a month for each additional site), the only expense you pay on your sales is the processing fee that you get

charged by Stripe (the same industry standard rates you get from all processors).

You end up with a simple, cheap, uncrashable custom store!

Chapter 1

Hi Ho, Hi Ho, it's Off to Work We Go!

Just in case you missed it in the Preface, I just want to quickly sum up who this tutorial was written for, and what is going to be covered. I've created this guide specifically with designers in mind because as someone who has been working in front-end design and development for years, I've always found that a lot of existing tutorials tend to avoid starting at first-principles. It seems like most people writing guides on how to accomplish bits and pieces of a development task assume that you have enough history to figure out all the other parts of getting your site set up. In the end though, that leads to needing to bounce around to a bunch of different tutorials, guides, stackoverflow threads, and other random sources of information, and I find having to do that exhausting...

So I decided that the best way to cover how to create a shop site would be to assume that you, the reader, is using an Apple computer running a recent version of OS X, have experience with design tools like Photoshop and Illustrator, have working knowledge or expertise with HTML and CSS, and have absolutely zero history in setting up a development environment to start working on an application.

If that isn't you, and you are only looking to see what the is involved in this approach, feel free to skip the sections on creating accounts, gem installation, and syncing / deploying. Just jump right into the parts about Jekyll, the product

builder, Stripe, and Cloudflare.

With that said...

This simple custom shopping site tutorial covers:

- setting up accounts for services you will need to use
- installing software and a development environment to use GitHub Pages to host your site's static files and provide version control for your application via Git
- the Jekyll framework and SASS to provide you with a convenient way to template your custom design, style your site's look and feel, and make updating content simple
- using Stripe to handle payment processing and emailing customers when their order is successfully placed
- setting up CloudFlare to provide an incredibly easy to configure SSL connection to your site.

This combination of services and technologies, you'll end up with a site that will cost you a flat $27 per month (assuming no one changes their prices) to serve to a basically limitless number of customers. If your product goes viral, not a problem! Pages will keep on loading and orders will be completed. Want to just test out a product? You can set up a page a matter of a couple hours and be selling products with a tiny overhead.

At the end of the tutorial is a separate section that covers some optional additions to the site if you want to get fancy and add in all the fun little details that make a web application rounded out and professional.

Let's get started!

Chapter 2

Dev Environment Intro, Create Accounts, Download Software

In order to get to the point where work can start on developing the actual site, you are first going to need to set up a number of accounts with a couple different services, download and install a few applications, and create a development environment. If you've never done this before, it might seem like a lot of work just to get going, but correctly setting up an effective dev environment is the foundation that all real software development is built on in the professional world.

2.1 What is a development environment?

A development environment (or dev environment) is the set of installed software that will run on your computer and will mimic the public server that will be building and running your application. This will allow you to work on your project by previewing changes in your browser locally (meaning not connected to the internet) without needing to constantly upload files to a server.

This is needed because unlike working with a simple flat HTML file, you

are going to be creating templates that define the outer HTML container, separate pages of content that get loaded into those templates, static content partials to allow you to not have to repeat elements like headers or footers, and data files that contain the details of your products. In order to see the site as it will appear to users, a server process will need to be running on your computer that will assemble all of those separate parts into actual pages that you can interact with.

2.2 The Terminal

Yes, you are going to need to use the terminal and work on the command line for this tutorial, but I promise, the command line stuff isn't hard even if it does look like a 1980s computer. You can use the built in OS X terminal found in the Utilities folder in your Applications directory, or you can also install a 3rd party terminal to get some additional features. I personally have been using iTerm for a long time, but it is just a personal preference thing.

When you see terminal commands in this tutorial, they will be presented like this:

```
$ command action
```

The $ sign is where your cursor is sitting waiting for you to start typing. In OS X and other Unix based operating systems, you'll see stuff in front of the $ that will identity the name of the computer, the folder that you are looking at, and the user account you are logged in as. Like this:

```
computername:directory username$ command action
```

To simplify things for this tutorial, I'll be using just the $ – just know that when you copy and past commands from the tutorial, don't include the '$'.

If you'd like more information about how the command line works and the different commands you can run there are a lot of resources on the internet. A great free resource is Mark Bates' book Conquering the Command Line

2.3 Setting up Stripe

The first thing that we are going to do is create a Stripe account. I just like to get this out of the way first, because if you have a problem setting up a payment processing service it is best to find out early.

Stripe is an easy to use payment processor that was build by a group of developers who grew fed up with the legacy payment processing options available and decided that they could do better. Stripe offers industry standard processing fees and an extremely well thought out API (application programming interface – the way that your application talks to their servers). You can read more about the service on the Stripe homepage.

Once you create your account, you will receive an activation email from Stripe asking you to confirm your account creation. After following the link, and confirming your email address by entering your password, you will be asked if you want to activate your account. Click the "Active account" button and enter your personal, business (if applicable), and banking information so that you get be set up and ready to start processing payments.

By default, Stripe will start your account in Test Mode (the toggle switch at the top left of your dashboard). This means that any payments you put through will not actually be charged when you use dummy credit card information. This lets you test your application without needing to deal with the chance of accidentally charging yourself a bunch of money. When you are ready to go live with your site, you'll have to throw the switch and put Stripe into Live mode, but more on that later – back to setting up your dev environment.

2.4 Your favorite text editor

To actually work on your project, you are going to need a text editor to write and edit the code. Personally, I'm a fan of Sublime text. It has a free trial version (the full license is $70), lots of bundles for add-ons, and has become a popular choice in the development community. You can also use an application like Textmate, the simple text editor that comes with the operating system, VIM, whatever – as long as it edits text it'll work.

I'd highly advise against using a what-you-see-is-what-you-get (WSIWYG) editor like Dreamweaver. We aren't going to be using any of it's live preview functionality, nor its drag and drop designing (which writes absolutely atrocious HTML), or its file management system. It really is a dinosaur and you should avoid it. Plus, you want to get that developer cred that comes from working with a cool text editor.

2.4.1 Sublime tweaks (optional)

If you do use Sublime, there are a couple little tweaks that you should do to make it awesome and get it ready for you to start developing this project. These are covered in the *Sublime Configuration Chapter 3* (which is completely optional – especially if you aren't using Sublime).

2.5 Congrats!

You made it this far! There is one more basic account creation and application installation, GitHub, left to set up. I've broken it out into it's own chapter though because it is also going to cover the initial creation of your project files and giving that its own chapter seemed like a good idea.

Chapter 3

Sublime Configuration (optional)

If you have decided to use Sublime, or you are already using the stock version of it with no modifications, let's spiff things up a little bit to make it prettier and more coding friendly. What this chapters covers is setting up Sublime to be able to be opened from the command line (which makes opening project files a lot easier), changing the way that Sublime handles tabs (to conform to the development standard), and installing a code highlighting package for Sass to make it easier to work with your stylesheets.

3.1 Sublime Command Line Launch

First let's set up Sublime to be able to open your project from the command line. This makes it much easier to get to your project and open an instance of Sublime focused only on your project files. After Sublime is installed into the Applications folder on OS X, open up your terminal app of choice, and paste this line in (as one line):

```
$ sudo mkdir -p /usr/local/bin && \
    sudo ln -s /Applications/Sublime\ Text\ 2.app/Contents/SharedSupport/bin/subl \
    /usr/local/bin/sublime
```

This will eventually allow you to open your project with the following command (which I'll repeat again when we are ready to start looking at project files).

```
$ sublime .
```

Yes, doing this will require you to use the **sudo** which allows the terminal to create folders and files in parts of your operating system that are designated as system directories. However, the **/usr/local/bin** directory is not used by OS X and is a conventional place to store files in a local development environment.

If you decide not to do the quick and dirty setup in a couple chapters, you will be installing Homebrew which is going to modify the permissions for the **usr/local/bin** directory and you won't need to use **sudo** again.

3.2 Sublime tabs to spaces, and tab sizing

Sublime ships with default settings that are a little different than the standard for document editing used in the development world. These days, it is normal for a text editor to use spaces in place of a tab character so that you avoid potentially including unintended characters in the code, and to set the tab size to be two spaces so that you aren't indenting long lines of text across the screen forcing ugly line wrapping.

Changing Sublime to follow these conventions will make any developer who looks at your code happy!

To configure the tab handling, open the Sublime default settings from the menu bar, ** Sublime Text 2 > Preferences > Settings - Default**. This will open a text file that you can edit to control the basic way that Sublime works.

Scroll down and make changes to the following values:

```
"tab_size": 2,
"translate_tabs_to_spaces": true,
```

3.3 Sublime Sass package

Last, we are going to install a package that will give you pretty code highlighting so that when you are working on your Sass/CSS styles all the elements and values will be nicely colored and easy to see.

With Sublime open, go to **View > Show Console**. This will open up the Sublime console and allow you to paste in code that can change how Sublime works. In this case we are going to add in a package manager that allows you to browse third party extensions to let you easily install the Sass plug-in.

Next, assuming you are using Sublime text 2, follow this link to the Sublime Package Controller, copy the code in the grey box, switch back to Sublime and paste it into the console that is open at the bottom of the Sublime application window:

After that runs, you'll need to close and then reopen Sublime. Then press p (command + shift + p) to open the Sublime plug-in manager and type:

```
install package
```

Hit enter to open the external package installation browser, and type in `Sass` to filter the packages.

Select the Sass plug-in (subtitle "Sass support for TextMate & Sublime Text 2") – it should be the first one. When we are ready to start working with Sass there will be one more step to make Sublime use this package by default for Sass files, but that will wait until we create the Sass file for the project.

If you are using Sublime text 3, you'll have to go the page and use the appropriate code for the Package Controller.

Chapter 4

Setting up Github and Git overview

4.1 GitHub Account

If you don't already have a GitHub account, now is the time to create one. GitHub is going to be storing your project files and also hosting them to the public through the GitHub Pages feature that is available to every GitHub user.

Go to the GitHub homepage, enter a username, your email address, and a password to start the sign-up process.

When you are presented with the choice of what type of account you are going to create, select the "Micro" plan. You can use the free account, but any code and assets that you create will be able to be accessed by anyone. Since we are creating a shopping site that is going to be handling payments from people it is probably for the best to keep all of your files private and secure.

Next, enter your payment information to start the $7 a month plan. When you have spare time, feel free to add in more information about you to complete your GitHub profile.

If you develop more applications and get more involved in the development world, you'll find that having a GitHub profile is a valuable thing when dealing with other developers, or if you are applying to get a job that requires development work.

4.2 GitHub Desktop Application

Now that your account has been created you should download the GitHub application for OS X. Using this desktop GitHub application will allow you to sync your site changes with GitHub in a user-friendly and visual way, avoiding using the command line (unless at some point you want to).

Visit the homepage for the OS X GitHub App, click the big download button, and drag the application into your 'Applications' folder after it finishes downloading to your computer.

When you start the application for the first time, you'll be presented with a guided setup to configure the application:

- On the first panel, you'll be asked to sign into GitHub using the credentials you just created. * Next, you'll be prompted to enter a name and email address to identify your code commits to other people if they join your repo. Before you leave this panel, be sure to click the 'Install Command Line Tools' button to install the Git specific terminal commands.
- On the last panel you won't see any local git repos, which is normal. Click done to finish.

Installing the command line tools will later let you play around with using git in the terminal if you are interested in the non-graphical user interface method of managing your project. I'll cover the command line method for handling your site repository as an optional sub-chapter later in this tutorial if you want to really dig into the terminal, but for now using the desktop application will make everything easier.

4.3 Git Introduction

At this point if you are unfamiliar with Git, you are probably asking, "what the hell is Git / GitHub and why did I just sign up for that?"

Git is a free and open-source version control system that was created by Linus Torvalds (the creator of the Linux open-source operating system) to help

him and the core contributors manage updating code. What Git allows you to do is to define a directory of files as a **'repository'** which Git will monitor in order to track changes to individual lines of code in each one of your project files. Unlike older version control systems that would require you to check out files that you are currently altering (preventing other people from accessing the file until you finish your changes), Git allows for you to have many different versions of the repository (repo) existing in different locations which then get synced together line by line to merge all the changes seamlessly (99% of the time) together.

GitHub is a service that provides you with a place to safely store a copy of your project repo to use as the master version. When working on your (and this works with any number of other people editing files in the repository) you are able to **'clone'** the repo from GitHub to a local computer, make changes to the files, **'commit'** to changes into the Git history for your project, and then sync the changes by **'pushing'** and **'pulling'** back to the main repo on GitHub. When you perform a sync of your repo, Git will update both the remote version on GitHub as well as your local files. This ensures that your local version is always up to date for you to make changes.

Unlike working with FTP, you'll never directly alter the site files that make up your application.

Additionally, Git supports a feature called **'branches'** which allow you to have multiple versions of a repository that can be kept separate from the main files that are publicly available to users. Branches are a really great way to make major changes to your code without having to worry that your public files are going to be affected. When you are ready to add the changes into the project, your updated files can be easily merged into the main branch. By default, GitHub Pages uses a specific branch, called **'gh-pages'**, on your project repo that the service monitors for updates. After you active GitHub Pages for your project you are going to set that branch as the main branch for your project since you aren't going to be using a traditional master branch.

If you want to read more on how Git works, there are a number of resources on the web to help you use and understand this incredibly powerful tool. One place to start is the Git Basics at git-scm.com

4.4 Creating your Git Repo

So let's start by creating your project repository!

In your browser, go to your GitHub dashboard by going to github.com which will automatically redirect you to your account's dashboard assuming you are signed in. If for some reason you signed out after creating your account, just click the 'Sign In' button at the top right.

In the page header, you'll see a '+' button next to your username on the right-hand side of the window – click that to bring up the drop-down, and select 'New Repository.'

You will be redirected then to the new repo page, and now you'll need to come up with a name for your project. If you don't have a perfect name in mind already don't worry, you can always change it later.

Make sure to click the radio button next to 'Private' in order to keep your project files from being publicly available, and also click the checkbox next to 'Initialize this repository with a README' (this adds a simple text file to the project just to start the repo). Click the 'Create Repository' button.

You just created your first repository! You will next be taken to the new repository's homepage which is the place where you can see all of your project files, browse through the change history, and configure your repository settings. Normally this view would be full of project files, but because you haven't created any other than the automatically generated README, it is going to look pretty empty… so let's use the GitHub Pages generator to get some project files in there!

4.5 Setting up GitHub Pages

When looking at your repo homepage, you'll see a vertical menu on the right-hand side of the page. At the bottom will be a an option labeled 'Settings'. Click that to go to the repo settings page and then scroll down to the box titled 'GitHub Pages'. Then click on the 'Automatic Page Generator' button to enable GitHub Pages for your project, as well as take you through a process to fill it with sample files to get started.

When the 'Create Page' form opens, you should first change what is in the 'Name' field to reflect what your site is going to be called (though if you don't know yet just leave it as the repo name). Ignore all the content in the 'Body' section for now, and if you have a Google Analytics ID ready for your new site you can enter it into the 'Google Analytics Tracking ID' field – if you don't have one ready, don't worry, it's easy to add later and I'll be covering how to set up traffic statistics in the optional *All the Bells and Whistles Chapter* **??**.

Next you'll be taken to a page to select a theme, but since we are going to be creating a completely custom experience you can just ignore the themes and just click the 'Publish' button taking you back to the repository home page. It might take a little bit of time before your GitHub Page is publicly ready, but that doesn't matter yet because you still have more to set up before you can start changing files. There's just one last little step to get things ready: click the 'Settings' option in the right-hand menu again, and in the dropdown next to 'Default Branch' label, select **'gh-pages'**. This is going to make the files in your GitHub Pages branch of the repository the default files that are on Github.

Go back to your repo homepage, and next we are going to get those files onto your computer.

4.6 Cloning your Repo

Near the bottom of the right-hand menu on your repo homepage you'll see a button that says 'Clone in Desktop,' click that and GitHub will open the desktop application and ask you where you'd like to save the project files.

You can save the files anywhere, but I prefer to save the files in a new directory called 'www' inside of my user folder, and have the GitHub application create my local repository inside that. Do create a 'www' folder, click down arrow next to the 'Clone As' field (unless you have the full folder view turned on by default), and then press the 'New Folder' button. Name it 'www' without the quotes, click 'Create'.

(After I've created the 'www' directory, I also like to drag the new folder to the sidebar as a favorite in OS X Finder so I can easily get to my project files. This is especially helpful when working with image files in other programs as

it's nice be able to quickly click on the 'www' directory, and then into a project. This ends up saving a great deal of time navigating through directories.)

Click the 'Clone' button and GitHub will add the repository name to the list of repos in the GitHub application, pull down your repository's files, and then show you the initial files that you created on the GitHub site.

Now that you've pulled the files down to your computer, you are going to need to finish setting up your local development environment so that you can start making changes.