HELLO WEB APP

LEARN HOW TO BUILD A WEB APP

Your next side project.

Your next lifestyle business.

Your next startup.

BY TRACY OSBORN

Hello Web App

Learn how to build a web app using Django and Python! Updated for Django 3.2.

Tracy Osborn

This book is for sale at http://leanpub.com/hellowebapp

This version was published on 2021-05-22

Tweet This Book!

Please help Tracy Osborn by spreading the word about this book on Twitter!

The suggested hashtag for this book is #hellowebapp.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#hellowebapp

For my family: Andrey, who is the best supporpoise anyone could ask for.

Westley, the 25lb biggest, baddest, most lovable cat.

Molly, who is the only kind of dog I'd ever want, except she peed on the bed last night. Dammit.

Contents

Introduction

Have you ever wanted to build something from scratch that other people could use? You could learn carpentry, knitting, or other physical crafts—but what about something for the web?

There are tons of tutorials and instructions for writing your first website using HTML and CSS, but building something that interacts with the user—a full, complete web application—might feel unachievable and out of reach.

The reality is that starting to build a web app is not as hard as you might think. Of course it's not easy, but today's tools can help a novice web developer create a basic web app in no time at all. It's only a matter of learning the basics and launching something real, and you'll be ready (and hopefully excited) to learn more.

I used to be a web designer with no appreciable programming experience. In fact, once upon a time, I did take some introductory computer science classes at my university. After a couple of semesters, I thought I hated programming (and especially with Java), which drove me to switch my field of study to Art. I vowed to never program again.

Fast forward again to when I was working as a web designer: I kept wishing certain web apps existed. I'm sure you know the feeling. Still convinced I hated programming, I tried finding a "technical cofounder" to help me launch an idea for a web app I had. It didn't work. I was back to where I started: an idea, no cofounder. I had two options: quit or finally try to write code again.

Friends introduced me to Python, a programming language that made way more sense to me than Java, and is simply nicer looking as well; and Django, a framework built on Python to help jumpstart the creation of web apps.

Slowly but surely, I built my startup, WeddingLovely. Over the next six years, I joined the startup accelerator 500 Startups and was accepted into the Designer Fund, raised money, and eventually made WeddingLovely into a mostly-bootstrapped profitable business.

It wasn't easy. The tutorials I found online all assumed previous programming knowledge. Crazy acronyms (like MVC) abounded, explanations only further confused me, and tutorials heavily relied on the command line, a tool friendly only to experienced programmers. As a web designer the results didn't feel "real" to me until I saw them on a website.

Hello Web App is what I wished had existed when I was learning to develop web apps. I wrote this book to be free of confusing explanations and unnecessary acronyms. I talk plainly, not academically, as if we're having a conversation between friends. This book doesn't teach "best practices"—it teaches the easiest way to launch a web app.

I hope you enjoy!

What We're Building

Python is a beautiful programming language. As a designer, I find the clean code and organization very appealing.

Django is a Python framework (like Ruby on Rails is for Ruby, another programming language you might have heard of). It is the most feature-complete and beginner-friendly web framework for Python: lots of useful utilities are built in and there is a massive amount of resources (tutorials as well as plugins) due to the size of the Django community.

But what exactly are we building?

Most tutorials start with a specific project. The official Django tutorial, for example, creates a polling app. But what if that tutorial subject doesn't interest you?

If you're like me, you would finish the tutorial but not feel any "ownership" over what you built because you essentially replicated another project. It's hard to relate and really understand what you're doing unless you feel involved. To that end, we're going to try something a little different here.

Hello Web App is going to walk you through building a generic "collection of things." However, this framework covers many different types of web apps you could build:

- A blog, which is a collection of posts.
- An online store, which is a collection of items you could buy.
- An online directory, which is a collection of people's profiles.
- … and so on and so forth.

What's written here is going to be generic and vanilla, and it's up to you to decide what exactly you're going to build using Hello Web

App. Pretty much the only thing you're going to change are the names of code bits, but the functionality will remain the same.

Some specific examples of what you could build using this book:

- A ratings website for a collection of things. Really love backpacking? You could create a website that shows your reviews for various pieces of equipment.
- A directory of people. This was my original project — I built a listing of custom wedding invitation designers. You could also do a listing of conference attendees, a list of awesome web people, such and so forth.
- An online store. There are a lot of solutions out there that help you set up a store without coding it, but it might be fun to build a store from the ground up to sell products.
- Or a blog, as mentioned above.

Take five minutes and think about a collection of objects that you're going to build using Hello Web App. Don't worry about scope just yet (we're getting to that); just find something you would be interested in working on.

What's your "collection of things" project?

MVP: Minimum Viable Product

Oh goodness, there's an acronym sneaking in, and I said I wouldn't do that.

Your MVP—Minimum Viable Product, a popular term in start-up land—is the minimum you need to build for your app to work and be useful to users.

Sometimes people get an idea for something they want to build and spend four years trying to perfect it. But there will always be another feature to add, another bug to fix, another thing to improve, all while you could be getting real people to use your app and give you feedback (**real** feedback) on how to improve.

Building your idea might seem really intimidating, especially when working with real customers. But having real customers will be an incredible motivation to work on your web app more.

For example, my first programming project mentioned before — today, it's chock full of features. There are free and paid accounts, using Stripe and PayPal to trigger recurring charges. There are a bunch of different ways to browse pages, such as by location, by budget, or by style. There's an API so other websites can integrate vendor listings into their websites powered by my app.

None of these features existed in the first version I built.

The only real features I needed for that first version were:

- A homepage.
- A profile page for each stationer in the directory.
- A basic search by location page.
- A form so a stationer could apply to join the directory.
- And static pages: About, Contact, etc.

It took me only six weeks from deciding to learn how to program to launch my first app. Two weeks later, it was profiled in a prominent design blog. Swamped by customers, my startup was born.

Even the simplest of web apps can grow into something big. Something you build now could become a business.

Take some time to write down your "collection of things" project idea, and then write down every awesome feature you think it should have. Then, circle only the ones that are really truly necessary. Make your web app as small and easy to launch as possible. With luck, the lessons of Hello Web App will be all you need to launch your app, and if not, you'll have only a few new concepts to learn before you can launch.

Prerequisites

HTML and CSS

This tutorial works best for those who have a solid understanding of HTML and CSS.

If you haven't worked with HTML before, there are tons of resources to help get you started with front-end development. Here are a few recommended resources:

- HTML and CSS: Design and Build Websites (http://hellobks.com/1)
- Learn to Code HTML & CSS (http://hellobks.com/2)
- Don't Fear the Internet: Basic HTML & CSS for Non-Web Designers (http://hellobks.com/3)

It is recommended and highly encouraged to be comfortable building a website using basic HTML and CSS before jumping into this book. If you're not, it won't take long to get there.

Python (just a bit)

Here's the big one!

"But this is a book on how to learn how to program. Why do I need Python knowledge?" you might be asking.

The best resources for learning the basic principles in Python have already been written and are online for free: no need for Hello Web App to reinvent the wheel.

My personal favorite: *Learn Python The Hard Way* (http://hellobks.com/4) by Zed Shaw, which contains a series of easy and well written Python exercises. Try to get through at least half. It shouldn't take very long.

Alternatively (or in addition), the video tutorial *Hands-On Intro to Python for Beginning Programmers* (http://hellobks.com/5) by Jessica McKeller is excellent.

Make sure you at least partially understand the following concepts (More info: http://hellobks.com/6):

- Variables
- If-statements
- For-loops
- Comments

If you don't feel like an expert, or even an intermediate, that's okay â€”Â these concepts will make more and more sense to you as you play around with programming. Once you feel like you have a basic grasp, we can start building our web app.

Suggestion: A Linux or Mac computer

Unfortunately, the Windows environment doesn't play as nicely with development and programming as Unix-based systems do (such as Linux and Mac operating systems.)

This doesn't mean you can't develop on Windows, and all Hello Web App instructions are written with both in mind. However, in general, life will be a lot easier on a Unix-based system, so if you have the option to use something other than a Windows environment, it's highly encouraged that you do so.

If you're using Windows and have any issues with Hello Web App's instructions while going through this book, check out our Windows resource page here: http://hellobks.com/7

Getting Started

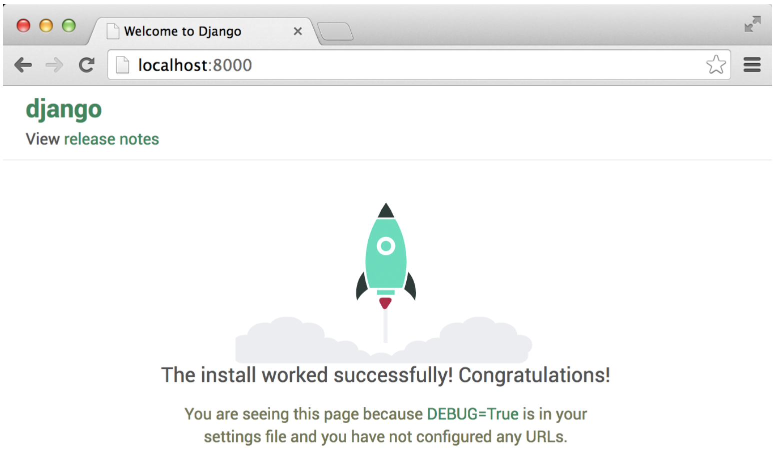
Writing HTML and viewing your results right away is easy: Just point your browser to your HTML (*.html*) file and voilà — your website gets displayed!

Not so with your Python (*.py*) files — your browser has no clue what to do with Python code. To start creating web apps in Django, we first need to install Python and Django on your computer (as well as a few other useful utilities, including a local web server that will interpret your Python code and deliver responses that your web browser can understand.)

This is the most complicated part of the process. Because the instructions for installing and setting up Python keep changing over time, they live online.

Hello Web App's Python and Django installation instructions can be found here: http://hellobks.com/8

Once you've finished, head back to your command line (make sure you're in your virtual environment and you're in the same directory as *manage.py*) and run `python manage.py runserver`. If everything worked correctly, you should see the Django congratulations page (congratulations!)



The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

What you should see if you correctly installed Python and Django and started the local web server.

As you go through the rest of the book, keep in mind that if you ever need to check that you have the correct code snippets copied into your app, all code in the book can be found on our GitHub code repository (change chapters by clicking the "branch" button): http://hellobks.com/9

We also have an online forum to discuss the book and resolve issues here: http://hellobks.com/10

Any other information (including installation instructions, tips, and resources) can be found on our main repository here: http://hellobks.com/11

Now get started building your app!

Setting Up Your Templates

Congrats, you've launched your Django web app! How do you actually make this look like a website? Note that the installation instructions didn't mention anything about templates, HTML, CSS, or any other files. Let's set that stuff up now.

Head back to your command line, and `cd` into your `collection` folder (the one with *models.py*). Using `mkdir`, create a "templates" directory, and then create *index.html* within it.

```
$ cd collection
collection $ mkdir templates
collection $ cd templates
collection/templates $ touch index.html
```

Open up *index.html* in your preferred code editor and add the necessary pieces to display a typical webpage. We're going to use HTML5 but only the bare bones for now — we can get fancy later.

index.html

```
<!doctype html>
<html>
    <head>
        <title>My Hello Web App Project</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <p>I am a basic website.</p>
    </body>
</html>
```

If you open up your browser and go to *http://localhost:8000*, you'll still see the same default page you saw before—none of the HTML above. We need to tell Django how to get to that file.

Adding a URL to `urls.py`

Django doesn't know what to do with the *index.html* file we just created. So head to *urls.py*, which is in the same folder as *settings.py* — we're going to associate the root directory (/) with the newly created *index.html*.

In *urls.py*'s entries area, find the `urlpatterns` line, remove the pre-existing comments, and add in the lines indicated:

urls.py

```
from django.contrib import admin
from django.urls import path

from collection import views

urlpatterns = [
    path('', views.index, name='home'),
    path('admin/', admin.site.urls),
]
```

There are essentially four parts to this new line: * `path()` encloses the path information. * `''` is where we indicate what route we're targeting after the domain. Since this is empty, we're targeting the homepage with our new template ("**www.example.com**"). For comparison, you can see that the second line (provided by Django) is targeting the admin path ("**www.example.com/admin**"). * `views.index` means that we'll use the index view in views.py in our app `collection` (imported at the top). We'll create this soon. * Last, `name='home'` is optional, but allows us to assign a name to this URL so we can refer to it in the future as "home". I'll explain how this ties in later.

I am using the "typical" urls.py layout here, but if it's more understandable to you, you can layout the URL definition like this:

urls.py

```python
urlpatterns = [
    path(
        route='',
        view=views.index,
        name='home',
    ),
    path('admin/', admin.site.urls),
]
```

This will help you keep track of which bit is what. I'm going to keep using the previous configuration in this book though, since it's how it's usually laid out in Django.

Apologies if URL configuration sounds complicated. The most you need to remember here is that there is path, then the view definition, and then we gave the URL a name. We can easily copy and paste from this template in the future for new URL entries.

Creating your first view

Now that we have a template and a URL entry, we need to tie them together, so that the URL will display the template. This is *views.py*'s job, which lives in the `collection` app.

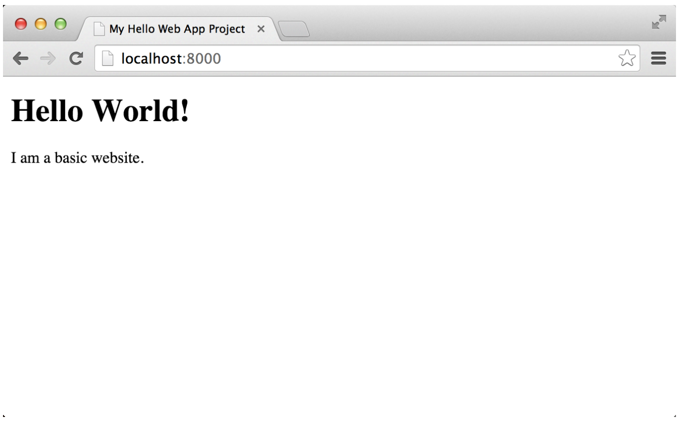
There are a ton of different ways to display a template simply in the views, and my favorite is Django's shortcut function called `render`. This is something you'll need to import, but Django anticipates that you'll use it and already has it added to the top of *views.py* for us.

views.py

```
from django.shortcuts import render

# Create your views here.
def index(request):
    # this is your new view
    return render(request, 'index.html')
```

Basically, `urls.py` will catch that someone wants the homepage and points to this piece of code, which will render the `index.html` template. Now open up your browser and check out the new homepage on *http://localhost:8000*. Woohoo!



Hello World!

I am a basic website.

Adding static files

We're now displaying the HTML file we created, but how do we get CSS styling in there? Unfortunately it's not as simple as creating a CSS directory and linking the stylesheets in our HTML file.

Let's go ahead and create the directory for static files now, which Django uses for files like style sheets. In your app (remember we named it `collection` — the folder that contains *models.py*), create a “static” directory, and within it, add in directories for CSS, Javascript, and images. In the CSS directory, add a blank *style.css* file.

```
$ cd collection
collection $ mkdir static
collection $ cd static
collection/static $ mkdir images
collection/static $ mkdir js
collection/static $ mkdir css
collection/static $ cd css
collection/static/css $ touch style.css
```

Head back into your *index.html* page. We’ll need to tell Django that there are static files used on this page, so at the top (like imports in *views.py* and *urls.py*) add `{% load staticfiles %}`. Why the `{%`? We’ll get to that soon.

Now that we can add static files to this template, add your CSS tag to the `<HEAD>` just like this:

index.html

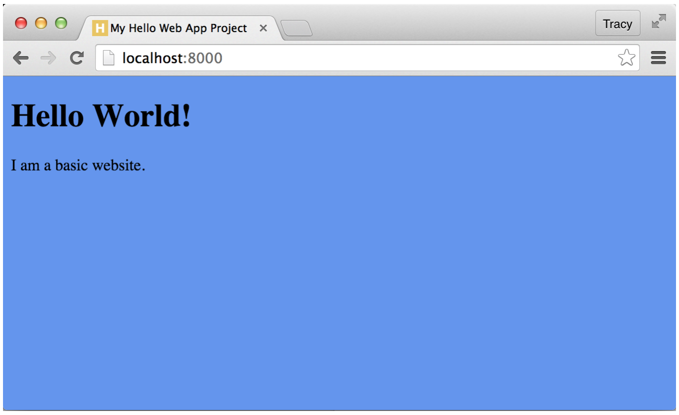
```
{% load static %}
<!doctype html>
<html>
    <head>
        <title>My Hello Web App Project</title>
        <link rel="stylesheet"
      href="{% static 'css/style.css' %}" />
    </head>
    <body>
        <h1>Hello World!</h1>
        <p>I am a basic website.</p>
    </body>
</html>
```

We want to avoid using relative paths, such as `href="../css/style.css"`. For optimization purposes, someday later you might want to move the static files of your project to a different server (such as Amazon’s *Simple Storage Service*, or *S3*), and by avoiding relative paths with Django's static files URL utility, your CSS path will always use what's defined in your settings and will magically work even if you change your static file's locations.

At this point, feel free to add some stylesheet declarations to your *style.css* file and check out *http://localhost:8000* again (run `python manage.py runserver` in your command line again if you need to do so).

style.css

```css
body {
    background-color: cornflowerblue;
}
```



Hello World!

I am a basic website.

Nice! Feel free to update your CSS more at this point. If it doesn't turn blue, try stopping and restarting your local server.

To link to any static file from a template, use the `{% static 'FILE-LOCATION/FILENAME.TYPE' %}` syntax, such as `{% static 'js/script.js' %}` or `{% static 'images/logo.jpg' %}`. Keep in mind you still need the `IMG` HTML tag when displaying images, for example: `<img src="{% static 'images/logo.jpg' %}" alt=""/>`

That said, don't worry about this within your CSS files — feel free to use relative paths because your static files should always be together anyways.

```css
h1 {
    /* for example... */
    background-image: url(../images/logo.png);
}
```

Now you can add static files and style your website! Don't forget to commit your work with git (review git on our git tips page here: http://hellobks.com/12).

Next up, Django has a bunch of awesome template utilities that'll elevate these static HTML files and make them a lot more interesting (and fun to work with).