HELLO WEB APP

INTERMEDIATE CONCEPTS

Learn the skills you need to create a successful, profitable web app.

BY TRACY OSBORN

Hello Web App: Intermediate Concepts

Add in the components you need to build a successful and profitable web app.

Tracy Osborn

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

This version was published on 2021-05-22

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

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

The suggested hashtag for this book is #hellowebappic.

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

#hellowebappic

Also By Tracy Osborn

Hello Web App

Really Friendly Command Line Intro

Really Friendly Git Intro

To the entire Django and Python community, since they've been nothing but welcoming and encouraging and I am truly grateful to work in such a wonderful industry.

And to my husband Andrey, who is, unfailingly, my biggest supporter and cheerleader. Without him, I wouldn't be where I am today.

(Quick high-five to my cat and dog.)

Contents

Introduction . . . . . **i**
Prerequisites iii
Our discussion forum iv

Creating a Contact Form and Working with Custom Forms **1**
Set up the URL 1
Set up the view 2
Set up the form 3
Create the template 4
Set up your local email server 5
Add the email logic 6
Create a template for your email 8
Improve the form (optional) 9
Set up our live email server (optional) 12
Things that could be improved 13
Your contact form is complete! 14

Introduction

Welcome to Hello Web App, the sequel!

A year ago, I wrote Hello Web App, a book that walks new programmers through building their own web app. It won't help you get a degree in Computer Science or Computer Engineering, nor is it a guide to getting a job as a developer or an engineer. Simply, the book helps people learn how to build web apps.

Readers can decide what's next after Hello Web App: learn more to become an engineer, hack on web apps as side-projects, or start building a lifestyle business or a startup (I did the last option and it turned out pretty awesome). Hello Web App is the next step in the learn-to-code revolution — beyond learning how to build a static website with HTML and CSS, we can build a fully functional web app and start working with customers.

All of this started after I taught myself — painfully — how to code half a decade ago (holy moly does time fly). I was a designer with an Art degree and loved doing front-end web development work. I had a lot of ideas for websites I wanted to build, but didn't want to hire someone to do the back-end development work for me.

After a few months of learning, I was able to cobble together a basic web app from several Django tutorials on

the web with copious amounts of help from my friends, and eventually launched a website. This website grew into my startup (and I the solo founder at the helm), which was accepted into a prominent startup accelerator and eventually raised funding.

During the years of refining my startup, I've learned more and more about web app development. The only tutorials available were frustratingly aimed at other developers — people who already knew how to code and who understood the jargon, references and side-notes. As I learned more development, I began to have mini-epiphanies: "Why the heck was it taught *that* way when it could be taught *this* way?" I realized that we needed a better way to teach web app development to those who didn't already know how to code. After years of waiting for this to be built and seeing no progress, I decided to write the book myself.

Hello Web App was Kickstarted in 2014 and launched on May 4th, 2015. Since then, thousands of folks have used the Hello Web App tutorial to create their first web app. The goal was to write a short, easy introduction to web app development, meaning the original book is the size of a small paperback. Hello Web App takes you from creating a project idea to launching your app on the internet so you can start working with real customers.

Consider this book, *Hello Web App: Intermediate Concepts*, as the whipped cream on top of a basic web app sundae. The chapters here don't rely on a chronological order, so you don't need to go directly from one chapter to the next

through the end of the book. Here, you can pick the chapter and concept you want to learn and just start building.

Also keep in mind that you don't need to have read the original Hello Web App for this book to be of use to you. Got a basic Django web app and want to take it to the next level? This book is for you.

This book is not going to have a lot of Computer-Science-y acronyms and engineering concepts. There are a lot of tutorials out there that will teach you Computer Science theory and best practices. Here, you'll learn how to do something from start to finish without a lot of asides and explanation about the why — just the how. And a *tiny* bit of theory.

We're building web apps, so we can create cool side projects — maybe even starting a lifestyle business or becoming the next startup. Learning web app development will open up so many doors for you.

Prerequisites

As mentioned before, this is a follow-up to the original Hello Web App but experience with the original book is **not required**. Do you have a basic Django web app and want to build some of the topics this book covers, like payment functionality? I got you.

One side-note: This book references the command-line command `touch` to create new files. Mac and Linux com-

puters have this ability natively, but unfortunately Windows computers don't. Either create the new files in your code-editor of choice by hand, or you can use *Git for Windows* http://hellowebapp.com/ic/01, which installs Git on your computer in addition to giving you a command line interface that lets you do UNIX commands such as `touch`.

Our discussion forum

If you have any issues while going through this book and Googling your question isn't giving you the answers you seek, check out the awesome Hello Web App discussion forum here: http://discuss.hellowebapp.com

Feel free to create a new topic if you're stuck and I'll pop in to help you within a few days (or some of the other awesome commentators may get back to you sooner). I also encourage you to share the app you've made for feedback, ask questions, or just say hi.

All right, let's get started!

Creating a Contact Form and Working with Custom Forms

In this walkthrough, we're going to build something relatively easy: a simple contact form where your users can enter their name, email address, and message, which will be emailed to you automatically by your website (with the user's email as the reply-to). In terms of the big picture, this will teach you how to create custom forms using Django, as so far in Hello Web App, we've only shown you how to create a `ModelForm`.

Set up the URL

Pretty much every new feature that will go into your web app will go through the same process: set up the URL, set up the logic, then set up the template. We're going to set up a simple page that lives at */contact/*. Add the new page to your *urls.py*:

urls.py

```
# make sure you're importing your views
# 'collection' is the name of my app, replace with yours
from collection import views

urlpatterns = [
    ...
    # new url definition
    path('contact/', views.contact, name='contact'),
```

Set up the view

Now in *views.py*, we need to start setting up the logic. Let's set it up to just display a form for now. Later on, we'll do the rest of the logic for after the form is submitted in a bit:

views.py

```
# add to the top
from .forms import ContactForm

# add to your views
def contact(request):
    form_class = ContactForm

    return render(request, 'contact.html', {
        'form': form_class,
    })
```

We're grabbing a form (which we haven't defined yet) and passing it over into the template (which we haven't created yet).

Set up the form

In Hello Web App, we went over creating forms with `ModelForms`, but skipped creating basic forms without a model. But it's just as simple to create custom forms!

In our *forms.py*, add the below form code:

forms.py

```
# make sure this is at the top if it isn't already
from django import forms

# our new form
class ContactForm(forms.Form):
    contact_name = forms.CharField()
    contact_email = forms.EmailField()
    content = forms.CharField(widget=forms.Textarea)
```

We're going to define the fields we want in our form, which will be just the contact's name, their email, and what they'd like to say to you.

All those form fields were grabbed from Django's form fields documentation (http://hellowebapp.com/ic/2), which is pretty easy to read to see what other fields are available. We're making all the form fields required, using an `EmailField` for the email so we can take advantage of the additional email formatting checks that Django provides, and making the "content" field a `Textarea`.

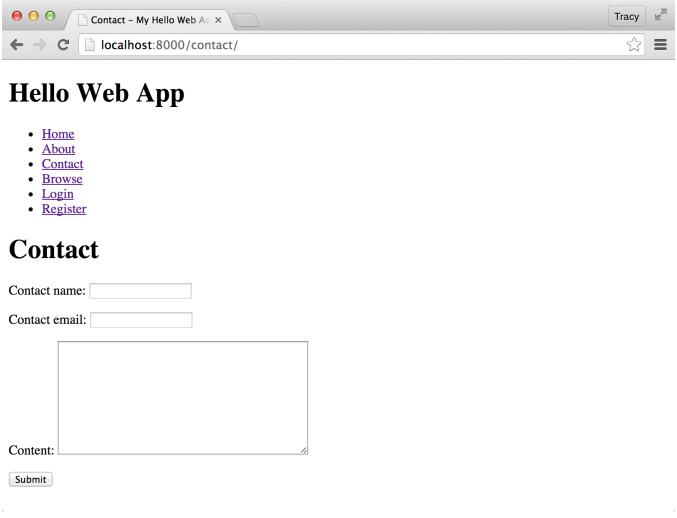
Create the template

Now we need to create the template page to display the contact form on our website. We're going to create the form using the form passed in from our view.

```
{% extends 'base.html' %}
{% block title %}Contact - {{ block.super }}{% endblock %}

{% block content %}
<h1>Contact</h1>
<form role="form" action="" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
</form>
{% endblock %}
```

At this point, we have all the pieces in place to display the form. Load */contact/* and check it out:



Nice! Now let's start adding the logic in the back-end to handle the information submitted by the user.

Set up your local email server

This will be redundant for you if you've already finished already the Hello Web App tutorial. In case you haven't, all you need to do to set up a local email server is add these lines to the bottom of your *settings.py*:

settings.py

```
EMAIL_BACKEND =
  'django.core.mail.backends.console.EmailBackend'
DEFAULT_FROM_EMAIL = 'testing@example.com'
EMAIL_HOST_USER = ''
EMAIL_HOST_PASSWORD = ''
EMAIL_USE_TLS = False
EMAIL_PORT = 1025
```

This tells Django to output the "email" to your console, where you ran your `python manage.py runserver` command. We'll see what this looks like in a second.

(This is only for local development — we'll get into email servers for your production web app at the end of this chapter.)

Add the email logic

Let's fill out the rest of the email logic. Here's the view from before, now filled in:

views.py

```
# new imports that go at the top of the file
from django.template.loader import get_template
from django.core.mail import EmailMessage
```

views.py

```
# our view
def contact(request):
    form_class = ContactForm

    # new logic!
    if request.method == 'POST':
        form = form_class(data=request.POST)

        if form.is_valid():
            contact_name = form.cleaned_data['contact_name']
            contact_email = form.cleaned_data['contact_email']
            form_content = form.cleaned_data['content']

            # Email the profile with the contact info
            template = get_template('contact_template.txt')

            context = {
                'contact_name': contact_name,
                'contact_email': contact_email,
                'form_content': form_content,
            }
            content = template.render(context)

            email = EmailMessage(
                'New contact form submission',
                content,
                'Your website <hi@example.com>',
                ['youremail@gmail.com'],
                headers = {'Reply-To': contact_email }
            )
            email.send()
            return redirect('contact')

    return render(request, 'contact.html', {
        'form': form_class,
    })
```

Phew, a lot of logic! If you read it from top to bottom, here's what's happening if the form was submitted:

- Apply the information from the form to the form class we set up before.
- Make sure that everything is valid (no missing fields, etc.)
- Take the form information and put it in variables.
- Stick the form information into a contact form template (which we will create momentarily).
- Create an email message using that contact template, and send the message.
- Redirect to our contact page (not ideal, we'll go into why below).
- Otherwise, just create the template with a blank form.

Create a template for your email

Before we can test our logic, we need to create an email template. Our email template is going to be simple, as it will just show the sections that our user filled out. Create a new file in your templates directory (`touch contact_template.txt`) and fill it in with the info below. Django will grab this file and fill it in using the context we set up in the view.

contact_template.txt

```
Contact Name:
{{ contact_name|striptags }}

Email:
{{ contact_email|striptags }}

Content:
{{ form_content|striptags }}
```

(We're using Django's template filter strip_tags to strip out HTML from the content. We need to be very careful with taking user input and presenting it as it was given. If we don't strip HTML, then a malicious user might put in some evil JavaScript in their input!)

Improve the form (optional)

In the screenshot of the form from before, we can see that the labels of the form aren't very "pretty" — for example, just saying "Contact name," which is very impersonal.

Django creates these names automatically from your field names, but we can set up our own pretty label names in the form definition in *forms.py*. To do so, update your code to the below:

forms.py

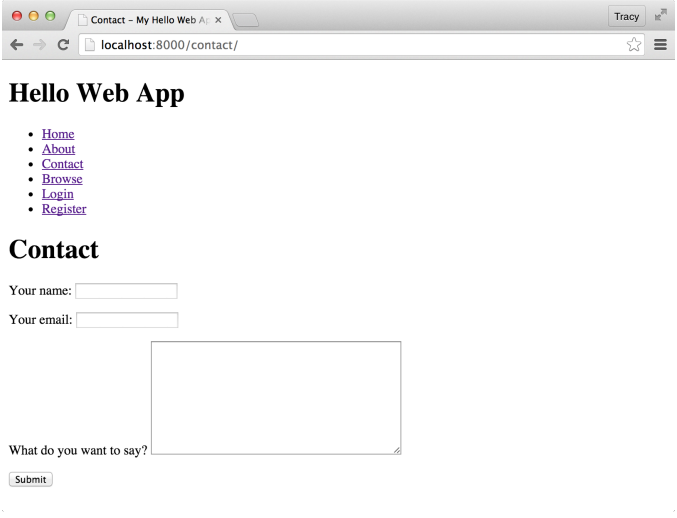
```
class ContactForm(forms.Form):
    contact_name = forms.CharField(required=True)
    contact_email = forms.EmailField(required=True)
    content = forms.CharField(
        required=True,
        widget=forms.Textarea
    )

    # the new bit we're adding
    def __init__(self, *args, **kwargs):
        super(ContactForm, self).__init__(*args, **kwargs)
        self.fields['contact_name'].label = "Your name:"
        self.fields['contact_email'].label = "Your email:"
        self.fields['content'].label = "What do you want to say?"
```

We've added the bit that starts with `__init__`, which might look a bit confusing. If you ignore the first two lines, the rest are pretty easy to read. We're just grabbing the relevant fields in our form and updating the label.

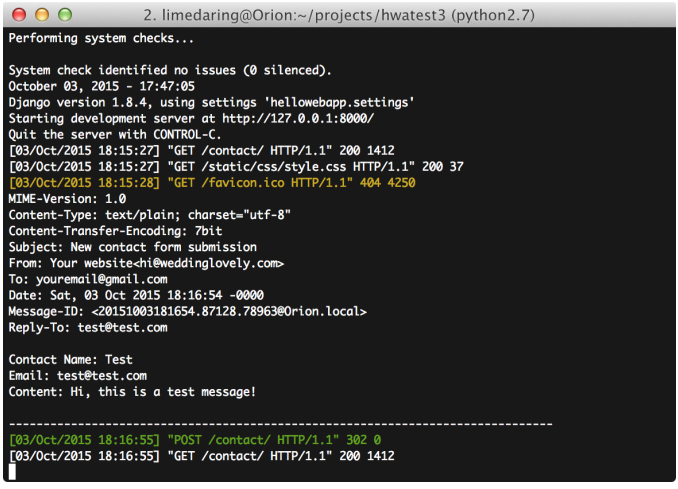
We can set more than just the label — we can also set the field as required, add help text, and other fields as well through `__init__`. You can see more information about updating form fields and attributes here in this excellent post: http://hellowebapp.com/ic/3

Once we've reloaded our form, we can see the new labels:



(Of course, this is minus any pretty CSS styling we need to do.)

Once we stick in some test information and submit the form, we can see the "email" in our command line:



Set up our live email server (optional)

The local email server will output "emails" to your local server (what's running in your command line), so you can confirm everything is working locally. But, when your web app is live, you obviously want those emails to actually land in your email inbox, rather than the server output.

You can do this by setting up something like Sendgrid (http://hellowebapp.com/ic/4) or Mandrill (http://hellowebapp.com/ic/5) — freemium email servers where you should just need to sign up for an account and set the details of your account in your *settings.py*.

Sendgrid has a great short walkthrough here: http://hellowebapp.com/ic/6. If you're at the point in Hello Web App where you've set up a production settings file, you can stick the email server stuff in there, and keep your local/test emails (using the Django console) in your normal *settings.py* file. This way you can "send emails" as you're developing your app, but you don't have to worry about going over the daily email limit that these email delivery products have in their freemium accounts.

Things that could be improved

I mentioned above that, upon successful form submission, you will be redirected to your app homepage. That would be really confusing to the user, because there is no success message. You have two options here:

- **Set up a separate template that just says "Success!" that users are redirected to after successful submission**. This is the easiest option, but adding these kind of templates tends to clutter up your templates directory.
- **Utilize the Django messages framework**. This is a better option. In your base template file, you can add a "messages" block, and then when you redirect to a page, you could pass along a message (e.g. an alert, an error, a warning, an info message, etc.) that will pop into the top of any page. It's what I use for my

production web apps. Chapter 6, *Setting up Django Messages for Alerts*, goes into this in detail.

Your contact form is complete!

You now have a working contact form that allows visitors to your web app to email you messages, and hopefully you learned some new skills about creating forms in Django and working with email. Congrats!