First News App Documentation

Release

Investigative Reporters and Editors

March 26, 2014

A step-by-step guide to publishing a simple news application.

This tutorial will walk you through the process of building an interactive data visualization from a structured dataset. You will get hands-on experience in every stage of the development process, writing Python, HTML and JavaScript and recording it in Git's version control system. By the end you will have published your work on the World Wide Web.

This guide was prepared for training sessions of Investigative Reporters and Editors (IRE) and the National Institute for Computer-Assisted Reporting (NICAR) by Ben Welsh. It debuted on February 27, 2014, at the Computer-Assisted Reporting Conference in Baltimore, MD.

- Code repository: https://github.com/ireapps/first-news-app
- Demonstration: http://ireapps.github.io/first-news-app/build/index.html
- Documentation: http://first-news-app.rtfd.org/
- Issues: https://github.com/ireapps/first-news-app/issues

CHAPTER 1

What you will make

This tutorial will guide you through the process of publishing an interactive database and map about the more than 60 people who died during the riots that swept Los Angeles for five days in 1992. You will repurpose the data from a Los Angeles Times application that accompanied a story released on the 20th anniversary of the riots.

A working example can be found at http://ireapps.github.io/first-news-app/build/index.html

CHAPTER 2

Prelude: Prerequisites

Before you can begin, your computer needs the following tools installed and working to participate.

1. A command-line interface to interact with your computer
2. A text editor to work with plain text files
3. Git version control software and an account at GitHub.com
4. Version 2.7 of the Python programming language
5. The pip package manager and virtualenv environment manager for Python

Note: Depending on your experience and operating system, you might already be ready to go with everything above. If so, move on to the next chapter. If not, don't worry. And don't give up! It will be a bit of a slog but the instructions below will point you in the right direction.

2.1 Command-line interface

Unless something is wrong with your computer, there should be a way to open a window that lets you type in commands. Different operating systems give this tool slightly different names, but they all have some form of it, and there are alternative programs you can install as well.

On Windows you can find the command-line interface by opening the "command prompt." Here are instructions for Windows 8 and earlier versions. On Apple computers, you open the "Terminal" application. Ubuntu Linux comes with a program of the same name.

2.2 Text editor

A program like Microsoft Word, which can do all sorts of text formatting like change the size and color of words, is not what you need. Do not try to use it below.

You need a program that works with simple "plain text" files, and is therefore capable of editing documents containing Python code, HTML markup and other languages without dressing them up by adding anything extra. Such programs are easy to find and some of the best ones are free, including those below.

For Windows, I recommend installing Notepad++. For Apple computers, try TextWrangler. In Ubuntu Linux you can stick with the pre-installed gedit text editor.

2.3 Git and GitHub

Git is a version control program for saving the changes you make to files over time. This is useful when you're working on your own, but quickly becomes essential with large software projects, especially if you work with other developers.

GitHub is a website that hosts git code repositories, both public and private. It comes with many helpful tools for reviewing code and managing projects. It also has some extra tricks that make it easy to publish web pages, which we will use later.

GitHub offers helpful guides for installing Git in Windows, Macs and Linux. You can verify it's installed from your command line like so:

```
# You don't have to type the "$" It's just a generic symbol
# geeks use to show they're working on the command line.
$ git --version
```

Once that's done, you should create an account at GitHub, if you don't already have one. It shouldn't cost you anything. The free plan is all that's required to complete this lesson.

2.4 Python

If you are using Mac OSX or a common flavor of Linux, Python is probably already installed and you can test to see what version, if any, is there waiting for you by typing the following into your terminal.

```
$ python -V
```

If you don't have Python installed (a more likely fate for Windows users) try downloading and installing it from here. In Windows, it's also crucial to make sure that the Python program is available on your system's `PATH` so it can be called from anywhere on the command line. This screencast can guide you through that process.

Python 2.7 is preferred but you can probably find a way to make most of this tutorial work with other versions if you futz a little.

2.5 pip and virtualenv

The pip package manager makes it easy to install open-source libraries that expand what you're able to do with Python. Later, we will use it to install everything needed to create a working web application.

If you don't have it already, you can get pip by following these instructions. In Windows, it's necessary to make sure that the Python `Scripts` directory is available on your system's `PATH` so it can be called from anywhere on the command line. This screencast can help.

Verify pip is installed with the following.

```
$ pip -V
```

The virtualenv environment manager makes it possible to create an isolated corner of your computer where all the different tools you use to build an application are sealed off.

It might not be obvious why you need this, but it quickly becomes important when you need to juggle different tools for different projects on one computer. By developing your applications inside separate virtualenv environments, you can use different versions of the same third-party Python libraries without a conflict. You can also more easily recreate your project on another machine, handy when you want to copy your code to a server that publishes pages on the Internet.

You can check if virtualenv is installed with the following.

```
$ virtualenv --version
```

If you don't have it, install it with pip.

```
$ pip install virtualenv
# If you're on a Mac or Linux and get an error saying you lack the right permissions, try it again as
$ sudo pip install virtualenv
```

If that doesn't work, try following this advice.

CHAPTER 3

Act 1: Hello Git

Start by creating a new development environment with virtualenv. Name it after our application.

```
# You don't have to type the "$" It's just a generic symbol
# geeks use to show they're working on the command line.
$ virtualenv first-news-app
```

Jump into the directory it created.

```
$ cd first-news-app
```

Turn on the new virtualenv, which will instruct your terminal to only use those libraries installed inside its sealed space. You only need to create the virtualenv once, but you'll need to repeat these "activation" steps each time you return to working on this project.

```
# In Linux or Mac OSX try this...
$ . bin/activate
# In Windows it might take something more like...
$ cd Scripts
$ activate
$ cd ..
```

Create a new Git repository.

```
$ git init repo
```

Jump into the repository.

```
$ cd repo
```

Visit GitHub and create a new public repository named `first-news-app`. Don't check "Initialize with README." You want to start with a blank repository.

Then connect your local directory to it with the following.

```
$ git remote add origin https://github.com/<yourusername>/first-news-app.git
```

Create your first file, a blank `README` with a Markdown file extension since that's the preferred format of GitHub.

```
# Macs or Linux:
$ touch README.md
# In Windows fire it up in your text editor right away:
$ start notepad++ README.md
```

Open up the README in your text editor and type something in it. Maybe something like:

```
My first news app
=================
```

Make sure to save it. Then officially add the file to your repository for tracking with Git's `add` command.

```
$ git add README.md
```

Log its creation with Git's `commit` command. You can include a personalized message after the `-m` flag.

```
$ git commit -m "First commit"
```

If this is your first time using Git, you may be prompted to configure you name and email. If so, take the time now. Then run the `commit` command above again.

```
$ git config --global user.email "your@email.com"
$ git config --global user.name "your name"
```

Now, finally, push your commit up to GitHub.

```
$ git push origin master
```

Reload your repository on GitHub and see your handiwork.

CHAPTER 4

Act 2: Hello Flask

Use pip on the command line to install Flask, the Python "microframework" we'll use to put together our website.

```
$ pip install Flask
```

Create a new file called `app.py` where we will configure Flask.

```
# Again, Macs and Linux:
$ touch app.py
# Windows:
$ start notepad++ app.py
```

Open `app.py` with your text editor and import the Flask basics. This is the file that will serve as your application's "backend," routing data to the appropriate pages.

```
from flask import Flask
app = Flask(__name__) # Note the double underscores on each side! You'll see them again.
```

Now configure Flask to make a page at your site's root URL, where we will publish the complete list of people who died during the riots using a template called `index.html`.

```
from flask import Flask
from flask import render_template
app = Flask(__name__)

@app.route("/")
def index():
    return render_template('index.html')
```

Return to your command-line interface and create a directory to store your templates in the default location Flask expects.

```
$ mkdir templates
```

Next create the `index.html` file we referenced in `app.py`. This is the HTML file where your will lay out your webpage.

```
# Macs and Linux:
$ touch templates/index.html
# Windows:
$ start notepad++ templates/index.html
```

Open it up in your text editor and write something clever.

```
Hello World!
```

Return to `app.py` and configure Flask to boot up a test server when you run it.

```
from flask import Flask
from flask import render_template
app = Flask(__name__)

@app.route("/")
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(
        host="0.0.0.0",
        port=8000,
        use_reloader=True,
        debug=True,
    )
```

Don't forget to save your changes. Then run `app.py` on the command-line and open up your browser to `http://localhost:8000` or `http://127.0.0.1:8000`.

```
$ python app.py
```

Now return to the command line and commit your work to your Git repository. (To get the terminal back up, you will either need to quit out of `app.py` by hitting `CTRL-C`, or open a second terminal and do additional work there. If you elect to open a second terminal, which is recommended, make sure to check into the virtualenv by repeating the `. bin/activate` part of *Act 1: Hello Git*. If you choose to quit out of `app.py`, you will need to turn it back on later by calling `python app.py` where appropriate.)

```
$ git add .
$ git commit -m "Flask app.py and first template"
```

Push it up to GitHub and check out the changes there.

```
$ git push origin master
```

CHAPTER 5

Act 3: Hello HTML

Start over in your `templates/index.html` file with a bare-bones HTML document.

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>Deaths during the L.A. riots</h1>
    </body>
</html>
```

Commit the changes to your repository, if only for practice.

```
$ git add templates/index.html
$ git commit -m "Real HTML"
$ git push origin master
```

Make a directory to store data files.

```
$ mkdir static
```

Download the comma-delimited file that will be the backbone of our application and save it there as `la-riots-deaths.csv`. Add it to your git repository.

```
$ git add static
$ git commit -m "Added CSV source data"
$ git push origin master
```

Open up `app.py` in your text editor and use Python's `csv` module to access the CSV data.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

csv_path = './static/la-riots-deaths.csv'
csv_obj = csv.DictReader(open(csv_path, 'r'))
csv_list = list(csv_obj)

@app.route("/")
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(
```

```
        host="0.0.0.0",
        port=8000,
        use_reloader=True,
        debug=True,
    )
```

Next pass the list to your template, `index.html`, so you can use it there.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

csv_path = './static/la-riots-deaths.csv'
csv_obj = csv.DictReader(open(csv_path, 'r'))
csv_list = list(csv_obj)

@app.route("/")
def index():
    return render_template('index.html',
        object_list=csv_list,
    )

if __name__ == '__main__':
    app.run(
        host="0.0.0.0",
        port=8000,
        use_reloader=True,
        debug=True,
    )
```

Make sure to save `app.py`. Then dump the data out in `index.html`. This is an example of Flask's templating language Jinja

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>Deaths during the L.A. riots</h1>
        {{ object_list }}
    </body>
</html>
```

If it isn't already running, return the command line, restart your test server and visit `http://localhost:8000` again.

Now we'll use Jinja to sculpt the data in `index.html` to create an HTML table that lists all the names.

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>Deaths during the L.A. riots</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
            </tr>
        {% for obj in object_list %}
            <tr>
```

```
                <td>{{ obj.full_name }}</td>
            </tr>
        {% endfor %}
        </table>
    </body>
</html>
```

Pause to reload your browser page. Next expand the table to include a lot more data.

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>Deaths during the L.A. riots</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
                <th>Date</th>
                <th>Type</th>
                <th>Address</th>
                <th>Age</th>
                <th>Gender</th>
                <th>Race</th>
            </tr>
        {% for obj in object_list %}
            <tr>
                <td>{{ obj.full_name }}</td>
                <td>{{ obj.date }}</td>
                <td>{{ obj.type }}</td>
                <td>{{ obj.address }}</td>
                <td>{{ obj.age }}</td>
                <td>{{ obj.gender }}</td>
                <td>{{ obj.race }}</td>
            </tr>
        {% endfor %}
        </table>
    </body>
</html>
```

Reload your page in the browser again to see the change. Then commit your work.

```
$ git add . # Using "." is a trick that will quickly stage *all* files you've changed.
$ git commit -m "Created basic table"
$ git push origin master
```

Next we're going to create a unique "detail" page dedicated to each person. Start by returning to `app.py` in your text editor and adding the URL that will help make this happen.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

csv_path = './static/la-riots-deaths.csv'
csv_obj = csv.DictReader(open(csv_path, 'r'))
csv_list = list(csv_obj)

@app.route("/")
def index():
```

```
    return render_template('index.html',
        object_list=csv_list,
    )

@app.route('/<number>/')
def detail(number):
    return render_template('detail.html')

if __name__ == '__main__':
    app.run(
        host="0.0.0.0",
        port=8000,
        use_reloader=True,
        debug=True,
    )
```

Create a new file in your templates directory called `detail.html` for it to connect with.

```
# Macs and Linux:
$ touch templates/detail.html
# Windows:
$ start notepad++ templates/detail.html
```

Put something simple in it with your text editor.

```
Hello World!
```

Then, if it's not running, restart your test server and use your browser to visit `http://localhost:8000/1/`, `http://localhost:8000/200/` or any other number.

```
$ python app.py
```

To customize the page for each person, we will need to connect the `number` in the URL with the `id` column in the CSV data file. First, return to `app.py` in the text editor and use Python to transform the data list we currently have there into a dictionary with each record's `id` as the key.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

csv_path = './static/la-riots-deaths.csv'
csv_obj = csv.DictReader(open(csv_path, 'r'))
csv_list = list(csv_obj)
csv_dict = dict([[o['id'], o] for o in csv_list])

@app.route("/")
def index():
    return render_template('index.html',
        object_list=csv_list,
    )

@app.route('/<number>/')
def detail(number):
    return render_template('detail.html')

if __name__ == '__main__':
    app.run(
        host="0.0.0.0",
```

```
        port=8000,
        use_reloader=True,
        debug=True,
    )
```

Then have the `detail` function connect the number from the URL with the corresponding record in the dictionary and pass it through the template.

```
import csv
from flask import Flask
from flask import render_template
app = Flask(__name__)

csv_path = './static/la-riots-deaths.csv'
csv_obj = csv.DictReader(open(csv_path, 'r'))
csv_list = list(csv_obj)
csv_dict = dict([[o['id'], o] for o in csv_list])

@app.route("/")
def index():
    return render_template('index.html',
        object_list=csv_list,
    )

@app.route('/<number>/')
def detail(number):
    return render_template('detail.html',
        object=csv_dict[number],
    )

if __name__ == '__main__':
    app.run(
        host="0.0.0.0",
        port=8000,
        use_reloader=True,
        debug=True,
    )
```

Now clear `detail.html` and make a new HTML document with a headline drawn from the data we've passed in from the dictionary.

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>{{ object.full_name }}</h1>
    </body>
</html>
```

Restart your test server and take a look at `http://localhost:8000/1/` again.

```
$ python app.py
```

Return to `index.html` and add a hyperlink to each detail page to the table.

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
```

```
        <h1>Deaths during the L.A. riots</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
                <th>Date</th>
                <th>Type</th>
                <th>Address</th>
                <th>Age</th>
                <th>Gender</th>
                <th>Race</th>
            </tr>
        {% for obj in object_list %}
            <tr>
                <td><a href="{{ obj.id }}/">{{ obj.full_name }}</a></td>
                <td>{{ obj.date }}</td>
                <td>{{ obj.type }}</td>
                <td>{{ obj.address }}</td>
                <td>{{ obj.age }}</td>
                <td>{{ obj.gender }}</td>
                <td>{{ obj.race }}</td>
            </tr>
        {% endfor %}
        </table>
    </body>
</html>
```

Restart your test server and take a look at `http://localhost:8000/`.

```
$ python app.py
```

In `detail.html` you can use the rest of the data fields to write a sentence about the victim and print out the summary that's been written in the data file.

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>
            {{ object.full_name }}, a {{ object.age }} year old,
            {{ object.race }} {{ object.gender|lower }} died on {{ object.date }}
            in a {{ object.type|lower }} at {{ object.address }} in {{ object.neighborhood }}.
        </h1>
        <p>{{ object.story }}</p>
    </body>
</html>
```

Reload `http://localhost:8000/1/` to see it. Then once again commit your work.

```
$ git add .
$ git commit -m "Created a detail page about each victim."
$ git push origin master
```

CHAPTER 6

Act 4: Hello JavaScript

Next we will work to make a map with every victim in `index.html` using the Leaflet JavaScript library. Start by importing it in your page.

```
<!doctype html>
<html lang="en">
    <head>
        <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />
        <script type="text/javascript" src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js?2"></sc
    </head>
    <body>
        <h1>Deaths during the L.A. riots</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
                <th>Date</th>
                <th>Type</th>
                <th>Address</th>
                <th>Age</th>
                <th>Gender</th>
                <th>Race</th>
            </tr>
        {% for obj in object_list %}
            <tr>
                <td><a href="{{ obj.id }}/">{{ obj.full_name }}</a></td>
                <td>{{ obj.date }}</td>
                <td>{{ obj.type }}</td>
                <td>{{ obj.address }}</td>
                <td>{{ obj.age }}</td>
                <td>{{ obj.gender }}</td>
                <td>{{ obj.race }}</td>
            </tr>
        {% endfor %}
        </table>
    </body>
</html>
```

Create an HTML element to hold the map and use Leaflet to boot it up and center on Los Angeles.

```
<!doctype html>
<html lang="en">
    <head>
        <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />
        <script type="text/javascript" src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js?2"></sc
```

```
    </head>
    <body>
        <div id="map" style="width:100%; height:300px;"></div>
        <h1>Deaths during the L.A. riots</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
                <th>Date</th>
                <th>Type</th>
                <th>Address</th>
                <th>Age</th>
                <th>Gender</th>
                <th>Race</th>
            </tr>
        {% for obj in object_list %}
            <tr>
                <td><a href="{{ obj.id }}/">{{ obj.full_name }}</a></td>
                <td>{{ obj.date }}</td>
                <td>{{ obj.type }}</td>
                <td>{{ obj.address }}</td>
                <td>{{ obj.age }}</td>
                <td>{{ obj.gender }}</td>
                <td>{{ obj.race }}</td>
            </tr>
        {% endfor %}
        </table>
        <script type="text/javascript">
            var map = L.map('map').setView([34.055, -118.35], 9);
            var mapquestLayer = new L.TileLayer('http://{s}.mqcdn.com/tiles/1.0.0/map/{z}/{x}/{y}.png
                maxZoom: 18,
                attribution: 'Data, imagery and map information provided by <a href="http://open.mapq
                subdomains: ['otile1','otile2','otile3','otile4']
            });
            map.addLayer(mapquestLayer);
        </script>
    </body>
</html>
```

Loop through the CSV data and format it as a GeoJSON object, which Leaflet can easily load.

```
<!doctype html>
<html lang="en">
    <head>
        <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />
        <script type="text/javascript" src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js?2"></sc
    </head>
    <body>
        <div id="map" style="width:100%; height:300px;"></div>
        <h1>Deaths during the L.A. riots</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
                <th>Date</th>
                <th>Type</th>
                <th>Address</th>
                <th>Age</th>
                <th>Gender</th>
                <th>Race</th>
            </tr>
```

```
        {% for obj in object_list %}
            <tr>
                <td><a href="{{ obj.id }}/">{{ obj.full_name }}</a></td>
                <td>{{ obj.date }}</td>
                <td>{{ obj.type }}</td>
                <td>{{ obj.address }}</td>
                <td>{{ obj.age }}</td>
                <td>{{ obj.gender }}</td>
                <td>{{ obj.race }}</td>
            </tr>
        {% endfor %}
        </table>
        <script type="text/javascript">
            var map = L.map('map').setView([34.055, -118.35], 9);
            var mapquestLayer = new L.TileLayer('http://{s}.mqcdn.com/tiles/1.0.0/map/{z}/{x}/{y}.png
                maxZoom: 18,
                attribution: 'Data, imagery and map information provided by <a href="http://open.mapq
                subdomains: ['otile1','otile2','otile3','otile4']
            });
            map.addLayer(mapquestLayer);
            var data = {
              "type": "FeatureCollection",
              "features": [
                {% for obj in object_list %}
                {
                  "type": "Feature",
                  "properties": {
                    "full_name": "{{ obj.full_name }}",
                    "id": "{{ obj.id }}"
                  },
                  "geometry": {
                    "type": "Point",
                    "coordinates": [{{ obj.x }}, {{ obj.y }}]
                  }
                }{% if not loop.last %},{% endif %}
                {% endfor %}
              ]
            };
            var dataLayer = L.geoJson(data);
            map.addLayer(dataLayer);
        </script>
    </body>
</html>
```

Add a popup on the map pins that shows the name of the victim.

```
<!doctype html>
<html lang="en">
    <head>
        <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />
        <script type="text/javascript" src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js?2"></sc
    </head>
    <body>
        <div id="map" style="width:100%; height:300px;"></div>
        <h1>Deaths during the L.A. riots</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
                <th>Date</th>
```

```
                <th>Type</th>
                <th>Address</th>
                <th>Age</th>
                <th>Gender</th>
                <th>Race</th>
            </tr>
        {% for obj in object_list %}
            <tr>
                <td><a href="{{ obj.id }}/">{{ obj.full_name }}</a></td>
                <td>{{ obj.date }}</td>
                <td>{{ obj.type }}</td>
                <td>{{ obj.address }}</td>
                <td>{{ obj.age }}</td>
                <td>{{ obj.gender }}</td>
                <td>{{ obj.race }}</td>
            </tr>
        {% endfor %}
        </table>
        <script type="text/javascript">
            var map = L.map('map').setView([34.055, -118.35], 9);
            var mapquestLayer = new L.TileLayer('http://{s}.mqcdn.com/tiles/1.0.0/map/{z}/{x}/{y}.png
                maxZoom: 18,
                attribution: 'Data, imagery and map information provided by <a href="http://open.mapq
                subdomains: ['otile1','otile2','otile3','otile4']
            });
            map.addLayer(mapquestLayer);
            var data = {
              "type": "FeatureCollection",
              "features": [
                {% for obj in object_list %}
                {
                  "type": "Feature",
                  "properties": {
                    "full_name": "{{ obj.full_name }}",
                    "id": "{{ obj.id }}"
                  },
                  "geometry": {
                    "type": "Point",
                    "coordinates": [{{ obj.x }}, {{ obj.y }}]
                  }
                }{% if not loop.last %},{% endif %}
                {% endfor %}
              ]
            };
            var dataLayer = L.geoJson(data, {
                onEachFeature: function(feature, layer) {
                    layer.bindPopup(feature.properties.full_name);
                }
            });
            map.addLayer(dataLayer);
        </script>
    </body>
</html>
```

Now wrap the name in a hyperlink to that person's detail page.

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

```
    <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />
    <script type="text/javascript" src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js?2"></sc
</head>
<body>
    <div id="map" style="width:100%; height:300px;"></div>
    <h1>Deaths during the L.A. riots</h1>
    <table border=1 cellpadding=7>
        <tr>
            <th>Name</th>
            <th>Date</th>
            <th>Type</th>
            <th>Address</th>
            <th>Age</th>
            <th>Gender</th>
            <th>Race</th>
        </tr>
    {% for obj in object_list %}
        <tr>
            <td><a href="{{ obj.id }}/">{{ obj.full_name }}</a></td>
            <td>{{ obj.date }}</td>
            <td>{{ obj.type }}</td>
            <td>{{ obj.address }}</td>
            <td>{{ obj.age }}</td>
            <td>{{ obj.gender }}</td>
            <td>{{ obj.race }}</td>
        </tr>
    {% endfor %}
    </table>
    <script type="text/javascript">
        var map = L.map('map').setView([34.055, -118.35], 9);
        var mapquestLayer = new L.TileLayer('http://{s}.mqcdn.com/tiles/1.0.0/map/{z}/{x}/{y}.png
            maxZoom: 18,
            attribution: 'Data, imagery and map information provided by <a href="http://open.mapq
            subdomains: ['otile1','otile2','otile3','otile4']
        });
        map.addLayer(mapquestLayer);
        var data = {
          "type": "FeatureCollection",
          "features": [
            {% for obj in object_list %}
            {
              "type": "Feature",
              "properties": {
                "full_name": "{{ obj.full_name }}",
                "id": "{{ obj.id }}"
              },
              "geometry": {
                "type": "Point",
                "coordinates": [{{ obj.x }}, {{ obj.y }}]
              }
            }{% if not loop.last %},{% endif %}
            {% endfor %}
          ]
        };
        var dataLayer = L.geoJson(data, {
            onEachFeature: function(feature, layer) {
                layer.bindPopup(
                    '<a href="' + feature.properties.id + '/">' +
```

```
                            feature.properties.full_name +
                        '</a>'
                    );
                }
            });
            map.addLayer(dataLayer);
        </script>
    </body>
</html>
```

Commit your map.

```
$ git add .
$ git commit -m "Made a map on the index page"
$ git push origin master
```

Open up `detail.html` and make a map there, focus on just that victim.

```
<!doctype html>
<html lang="en">
    <head>
        <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />
        <script type="text/javascript" src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js?2"></sc
    </head>
    <body>
        <div id="map" style="width:100%; height:300px;"></div>
        <h1>
            {{ object.full_name }}, a {{ object.age }} year old,
            {{ object.race }} {{ object.gender|lower }} died on {{ object.date }}
            in a {{ object.type|lower }} at {{ object.address }} in {{ object.neighborhood }}.
        </h1>
        <p>{{ object.story }}</p>
        <script type="text/javascript">
            var map = L.map('map').setView([{{ object.y }}, {{ object.x }}], 16);
            var mapquestLayer = new L.TileLayer('http://{s}.mqcdn.com/tiles/1.0.0/map/{z}/{x}/{y}.png
                maxZoom: 18,
                attribution: 'Data, imagery and map information provided by <a href="http://open.mapq
                subdomains: ['otile1','otile2','otile3','otile4']
            });
            map.addLayer(mapquestLayer);
            var marker = L.marker([{{ object.y }}, {{ object.x }}]).addTo(map);
        </script>
    </body>
</html>
```

Commit that.

```
$ git add .
$ git commit -m "Made a map on the detail page"
$ git push origin master
```

CHAPTER 7

Act 5: Hello Internet

In this final act, we will publish your application to the Internet using Frozen Flask, a Python library that saves every page you've made with Flask as a flat file that can be uploaded to the web. This is an alternative publishing method that does not require you configure and host an full-fledged Internet server.

First, use pip to install Frozen Flask from the command line.

```
$ pip install Frozen-Flask
```

Create a new file called `freeze.py` where we will configure what pages it should convert into flat files.

```
# Mac and Linux:
$ touch freeze.py
# Windows:
$ start notepad++ freeze.py
```

Use your text editor to write a basic Frozen Flask configuration.

```
from flask_frozen import Freezer
from app import app
freezer = Freezer(app)

if __name__ == '__main__':
    freezer.freeze()
```

Now run it from the command line, which will create a new directory called `build` filled with a set of flattened files.

```
$ python freeze.py
```

Use your browser to open up one of the local files in `build`, rather that visit the dynamically generated pages we created at `localhost`.

You will notice that the default Frozen Flask configuration only flatted out `index.html`, and not all your detail pages our template could generate using the data file.

To flatten those, again edit `freeze.py` to give it the instructions it needs to make a page for every record in the source CSV.

```
from flask_frozen import Freezer
from app import app, csv_list
freezer = Freezer(app)

@freezer.register_generator
def detail():
    for row in csv_list:
        yield {'number': row['id']}
```

```
if __name__ == '__main__':
    freezer.freeze()
```

Run it again from the command line and notice all the additional pages it made in the `build` directory. Try opening one in your browser.

```
$ python freeze.py
```

Commit all of the flat pages to the repository.

```
$ git add .
$ git commit -m "Froze my app"
$ git push origin master
```

Finally, we will publish these static files to the web using GitHub's Pages feature. All it requires is that we create a new branch in our repository called `gh-pages` and push our files up to GitHub there. Keep in mind there are many other options for publishing flat files, ranging from Dropbox to Amazon's S3 service.

```
$ git checkout -b gh-pages # Create the new branch
$ git merge master # Pull in all the code from the master branch
$ git push origin gh-pages # Push up to GitHub from your new branch
```

Now wait a minute or two, then visit `http://<yourusername>.github.io/first-news-app/build/index.html` to cross the finish line.