First News App Documentation

Publicación

Investigative Reporters and Editors

27 de September de 2017

Índice general

Una guia paso a paso para publicar una aplicación sencilla de noticias.

Este manual va a guiarte pro el proceso de construir una visualización interactiva de un conjunto de datos estructurado. Van a practicar sobre cada paso del proceso de desarrollo, escribiendo Python, HTML y Javascript y guardandolo en el sistema de control de versiones Git.

Al final van a tener el trabajo de ustedes publicarlo en Internet.

Esta guia es basada en las sesiones de *Ben Welsh <http://palewi.re/who-is-ben-welsh/>* para Investigative Reporters and Editors (IRE) y el National Institute for Computer-Assisted Reporting (NICAR) by _. el February 27, 2014. Fue traducida y adaptada para periodistas españoles, a presentarse durante las Jornadas de Periodismo de Datos, en Madrid y Barcelona 2014, por *Gabriela Rodriguez <http://www.gabrielarodriguez.org>*.

- Repositorio de código: https://github.com/gabelula/first-news-app-barna
- Demo: http://gabelula.github.io/espanaenllamas/build/index.html
- Documentación: http://first-news-app-spanish.rtfd.org/
- Problemas: https://github.com/gabelula/first-news-app-barna/issues

CAPÍTULO 1

¿Qué vamos a hacer?

Este tutorial te guiara por el proceso de publicar una base de datos y mapa interactivo sobre los incendios en España. Se utilizará los datos de *España En Llamas <http://www.espanaenllamas.es>*.

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

CAPÍTULO 2

Preludio: Pre-requisitos

Antes de empezar, tu computadora va a necesitar las siguientes herramientas instaladas y trabajando para poder participar.

1. Una linea de comandos para interactuar con tu computadora
2. Un editor de texto para trabajar en archivos de texto plano
3. Git software de control de versiones y una cuenta en GitHub.com
4. Version 2.7 de Python lenguaje de programación
5. El pip manejador de paquetes de software y virtualenv manejador de ambientes para Python

Linea de Comandos

A menos que algo este mal con tu computadora, debería haber una forma de abrir una ventana que te deje tipear comandos. Diferentes sistemas operativos le dan un nombre diferente a esta herramienta, pero todos usan alguna forma de terminal, y también hay programas que puedes instalar.

En Windows puedes encontrar una interfase como la linea de comandos al abrir el "prompt de comandos". Aqui hay una guia para Windows 8 y otras versiones en <http://windows.microsoft.com/en-us/windows-vista/open-a-command-prompt-window>‘_. En computadoras Apple, tu puedes abrir la aplicación "Terminal". Ubuntu Linux viene con un programa del mismo nombre.

Editor de Texto

Un programa como Microsoft Word, que puede cambiar el color y tamaño del color de palabras, no es lo que necesitas.

Necesitas un software que trabaje con archivos de texto plano <https://en.wikipedia.org/wiki/Text_file>‘_, y en el que se pueda editar documentos que tengan código Python, HTML y de otros lenguajes sin que cambie la extensión del archivo automaticamente o agregue ninguna otra cosa extra. Estos programas son faciles de encontrar y muchos son de código libre.

Para Windows, yo recomiendo instalar Notepad++. Para Mac, pueden probar *TextWrangler <http://www.barebones.com/products/textwrangler/download.html>* o Sublime. En Ubuntu Linux pueden abrir gedit que seguramente ya este instalado.

Git y GitHub

Git es un software de control de versiones para guardar los cambios que se hacen a los archivos a lo largo del tiempo. Esto es útil cuando esten trabajando sólos pero rápidamente se convierte en imprescindible cuando trabajando en proyectos de software grandes, especialemtne cuando se trabaja con otros programadores.

GitHub es un sitio web que guarda repositorios de código, tanto públicos como privados. Viene con muchas herramientas para revisar código y manejar proyectos. También incluye trucos extras que hacen fácil la publicación gratuita de páginas web.

GitHub tiene varios tutoriales para instalar Git en Windows, Macs y Linux. Puedes comprobar que esta instalado en tu linea de comandos con:

```
# No tienes que tipear "$" . Es sólo un simbolo generico
# que usamos para mostrar que estamos trabajando en la linea de comandos.
$ git --version
```

Una vez que has hecho esto, deberias crear una cuenta en GitHub, si aún no tienes una. Hay un plan gratuito con el que podemos trabajar sin problemas.

Python

Si usas Mac OSX o algún tipo de Linux, Python está seguramente ya instalado y tu puedes mirar que versión tienes instaladas, tipeando en la terminal lo siguiente.

```
$ python -V
```

Si no tienes Python instalado (algo casi seguro para quienes usan Windows) intenta descargarlo e instalarlo desde aqui. En Windows, es muy importante de estar seguros que Python esta disponible desde el `PATH` de tu sistema para que puedas llamarlo desde cualquier lugar en la linea de comandos. Aqui hay un screencast en ingles que puede guiarte en ese proceso.

Python 2.7 es la versión que estamos usando pero seguramente puedes usar este tutorial con otras versiones también.

pip y virtualenv

El manejador de paquetes pip te facilita la instalación de librerias open-source que te ayuden al trabajo que puedas hacer con Python. Lo vamos a usar para instalar todo lo que necesitemos para crear una aplicación web.

Si aún no lo tienes instalado, puedes tener pip siguiendo estas instrucciones. En Windows, hay que asegurarse de tener el directorio de `Scripts` de Python este disponible en tu `PATH` del sistema, para que pueda ser llamado desde cualquier lugar en la linea de comandos. Este screencast (en ingles) puede ayudar.

Comprobar que pip este instalado con lo siguiente.

```
$ pip -V
```

El manejador de ambientes de trabajo virtualenv hace posible crear un lugar aislado de tu computadora donde estan todas las herramientas que usas para construir tu aplicación.

No es obvio porque podrais necesitar esto, pero se hace más importante cuando necesitas manejar diferentes herramientas para diferentes proyectos en sólo una computadora. al desarrollar tus aplicaciones dentro de ambientes virtuales separados, puedes usar diferentes versiones de las mismas librerias sin tener ningún conflicto. También permite que facilmente recreen el proyecto en otra máquina, lo que se hace muy conveniente cuando se quiere copiar el código a un servidor para publicarlo en Internet.

Puedes comprobar que virtualenv este instalado con lo siguiente.

```
$ virtualenv --version
```

Si aún no lo tienes, instalalo con pip.

```
$ pip install virtualenv
# Si estas en una Mac o Linux y sale un error diciendo que hay un problema de permisos, intentalo co
$ sudo pip install virtualenv
```

Si eso no funciona, intenta seguir este consejo.

CAPÍTULO 3

Act 1: Hola Git

Empezar creando un ambiente nuevo de desarrollo con virtualenv. Le puedes poner cualquier nombre pero es práctico que se llame igual que tu aplicación.

```
# No tienes que tipear "$" . Es sólo un simbolo generico
# que usamos para mostrar que estamos trabajando en la linea de comandos.
$ virtualenv first-news-app
```

Ir a la carpeta que se creo.

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

Activar el nuevo virtualenv, que va a hacer que tu terminal sólo utilize las librerias que estan instaladas dentro de ese ambiente. Sólo hay que crear el virtualenv una vez, pero hay que repetir el proceso de activación cada vez que vas a trabajar en el proyecto.

```
# En Linux ó Mac OSX haz esto...
$ . bin/activate
# En Windows debería ser algo como...
$ cd Scripts
$ activate
$ cd ..
```

Crear un nuevo repositorio de GIT.

```
$ git init repo
```

Ir al repositorio que se acaba de crear.

```
$ cd repo
```

Ir a GitHub y crear un repositorio publico llamado `first-news-app`. No marcar "Initialize with README." Vamos a comenzar con un repositorio vacio.

Y conectar el directorio local con el repositorio en github con lo siguiente.

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

Crea tu primer archivo y llámalo `README` con extension de archivo Markdown pues ese es el formato preferido de GitHub <https://help.github.com/articles/github-flavored-markdown>`_.

```
# Macs ó Linux:
$ touch README.md
# En Windows abrirlo con un editor de texto:
$ start notepad++ README.md
```

Abrir README con un editor de texto y tipear cualquier cosa. Puede ser algo por el estilo de:

```
Mi primera aplicación de noticias
=================================
```

Recuerda de guardarlo. Luego agrega el archivo a tu repositorio mediante el comando de Git `add`.

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

Y confirma que quieres guardar ese cambio con el comando `commit`. Puedes incluir un mensaje con la opción `-m`.

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

Si esta es la primera vez que usas Git, puede ser que te pregunte por tu nombre y correo. Si lo hace, toma el tiempo ahora para agregarlo. Corre el comando `commit` nuevamente.

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

Ahora, publica lo que commiteaste a GitHub.

```
$ git push origin master
```

Refresca el repositorio en la web de GitHub y veras lo que acabas de enviar.

CAPÍTULO 4

Act 2: Hola Flask

Usar pip en la linea de comandos para instalar Flask, el "microframework" de Python que vamos a usar para crear nuestro sitio web.

```
$ pip install Flask
```

Crear un nuevo archivo llamado `app.py` donde vamos a configurar Flask.

```
# En Macs y Linux:
$ touch app.py
# Windows:
$ start notepad++ app.py
```

Abrir `app.py` con tu editor de texto e importar lo básico de Flask. Este es el archivo que va a servir como el "backend" de tu aplicación, redirigiendo datos a las páginas apropiadas.

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

Ahora configura Flask para hacer una página que cargue en la URL raíz de tu web, donde se va a publicar una lista completa de los incendios usando un template llamado `index.html`. Le llamamos template al esqueleto en html donde se va a cargar el contenido desde nuestros datos.

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

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

Volver a la linea de comandos y crear una carpeta donde guardar tus templates. Flask espera, por defecto, que esta carpeta se llame templates.

```
$ mkdir templates
```

Ahora crear el archivo `index.html` que nombramos en `app.py`. Este es el archivo HTML donde vamos a colocar nuestra página web.

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

Abrirlo en tu editor de texto y escribir cualquier cosa.

```
¡Hola Mundo!
```

Volver a editar `app.py` y configurar Flask para levantar un servidor de prueba cuando lo corramos.

```
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,
    )
```

Recuerda guardar los cambios. Entonces corre `app.py` en la linea de comandos y abrir el navegador en `http://localhost:8000` o `http://127.0.0.1:8000`.

```
$ python app.py
```

Ahora vuelve a la linea de comandos y commitea tu trabajo al repositorio Git. (Para volver a tener la linea de comandos disponibles, vas a tener que terminar `app.py` apretando `CTRL-C`, o abrir una segunda terminar y seguir trabajando alli. Si eliges abrir una segunda terminar, recuerda que necesitas activar nuevamente el ambiente virtual con `. bin/activate` part of *Act 1: Hola 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 y el primer template"
```

Publicalo en GitHub y mira los cambios alli.

```
$ git push origin master
```

CAPÍTULO 5

Act 3: Hola HTML

Editar nuevamente el archivo `templates/index.html` agregandole un esqueleto de un archivo HTML.

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>Incendios en España desde 2001 hasta 2011</h1>
    </body>
</html>
```

Commitear los cambios al repositorio.

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

Crear un directorio, donde guardaremos los datos.

```
$ mkdir static
```

Descargar el archivo que será la base de nuestra aplicación y guardarla como `incendios.csv`. Agregarlo al repositorio.

```
$ git add static
$ git commit -m "Agregando la fuente de datos en CSV"
$ git push origin master
```

Abrir nuevamente `app.py` en tu editor de texto y usar el modulo de python `csv` para acceder a los datos del CSV.

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

csv_path = './static/incendios.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,
    )
```

El próximo paso es pasarle la lista al template, `index.html`, para poder usarlo alli.

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

csv_path = './static/incendios.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,
    )
```

Recuerdo guardar `app.py`. Y colocar los datos directamente en `index.html`. Este es un ejemplo del lenguaje de template de Flask llamado Jinja

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>Incendios en España del 2001 al 2011</h1>
        {{ object_list }}
    </body>
</html>
```

Si no está corriendo, volver a la linea de comandos y reiniciar el servidor de pruebas y visitar nuevamente `http://localhost:8000`.

Vamos a usar Jinja para mostrar los datos en `index.html` creando una HTML de tag table para listar todos los nombres. Algunos datos los tenemos que descodificar pues estamos usando castellano. Para eso usamos el encoding *UTF-8 <http://es.wikipedia.org/wiki/UTF-8>*.

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>Incendios en España del 2001 al 2011</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
            </tr>
```

```
        {% for obj in object_list %}
            <tr>
                <td>{{ obj['COMUNIDAD'].decode('UTF-8') }}</td>
            </tr>
        {% endfor %}
        </table>
    </body>
</html>
```

Refrescar la página en el navegador. Ahora vamos a colocar el resto de los datos que nos parezcan interesantes.

```
<!doctype html>
<html lang="en">
    <head></head>
    <body>
        <h1>Incendios en España del 2001 al 2011</h1>
        <table border=1 cellpadding=7>
            <tr>
              <th>Superficie Forestal Quemada</th>
              <th>Muertos</th>
              <th>Heridos</th>
              <th>Comunidad</th>
              <th>Causa</th>
              <th>Perdidas</th>
            </tr>
        {% for obj in object_list %}
            <tr>
              <td>{{ obj['SUPQUEMADA'] }}</td>
              <td>{{ obj['MUERTOS'] }}</td>
              <td>{{ obj['HERIDOS'] }}</td>
              <td>{{ obj['COMUNIDAD'].decode('UTF-8') }}</td>
              <td>{{ obj['CAUSA'].decode('UTF-8')  }}</td>
              <td>{{ obj['PERDIDAS'] }}</td>
            </tr>
        {% endfor %}
        </table>
    </body>
</html>
```

Refrescar la página en el navegador nuevamente para ver los cambios. Luego commitea tu trabajo.

```
$ git add . # Usar "." es un truco que va a meter *todos* los archivos que has cambiado a estado prep
$ git commit -m "Creó tabla basica"
$ git push origin master
```

El siguiente paso es crear una página "detail" para cada incendio. Editar nuevamente `app.py` y agregar la URL que va a levantar la página de detalles.

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

csv_path = './static/incendios.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,
    )
```

Crear un archivo nuevo, en la carpeta templates, llamado `detail.html`.

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

Colocar algo simple en el archivo con tu editor de texto.

```
¡Hola Mundo!
```

Reiniciar el servidor de prueba y usar un navegador para visitar `http://localhost:8000/1/`, `http://localhost:8000/200/` o cualquier otro número.

```
$ python app.py
```

Para tener información especifica de cada incendio, debemos conectar el `numero` en la URL con la columna `id` en el archivo de datos CSV. Primero, editar `app.py` en el editor de texto y usa Python para convertir la lista de datos que tenemos hasta ahora en un diccionario donde cada `id` del registro es la llave.

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

csv_path = './static/incendios.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,
    )
```

Luego hacer que la funcion `detail` conecte el numero desde la URL con el registro correspondiente en el diccionario y que se lo pase al template `detail.html`.

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

csv_path = './static/incendios.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,
    )
```

Ahora, borra todo lo el contenido que tenias en el archivo `detail.html` y coloca el código HTML para que tenga un titulo con los datos que le pasamos desde el diccionario en `app.py`.

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

Reiniciar tu servidor de pruebas y mira nuevamente en el navegador `http://localhost:8000/1/`.

```
$ python app.py
```

Vuelve a editar `index.html` y agregale un enlace a cada página de detalle.

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

```
        <h1>Incendios en España entre 2001 y 2011</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>
```

Reinicia tu servidor de pruebas y mira `http://localhost:8000/`.

```
$ python app.py
```

En `detail.html` puedes usar el resto de los campos de datos para escribir una oración sobre el incendio.

```
<!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>
```

Refrescar `http://localhost:8000/1/` para verla. Y commitea nuevamente tu trabajo.

```
$ git add .
$ git commit -m "Creo una página para cada incendio."
$ git push origin master
```

Act 4: Hola JavaScript

Ahora vamos a trabajar en hacer un mapa con cada incendio en `index.html` usando la libreria de Javascript Leaflet. Primero hay que importarla en la página.

```
<!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>Incendios en España desde 2001 hasta 2011</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>
```

Crear un elemento HTML que llevara el mapa y usara Leaflet para levantarlo y centrarlo en España.

```
<!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>Incendios en España desde 2001 hasta 2011</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: 'Datos e imagenes de mapas servidos por <a href="http://open.mapquest.co
                subdomains: ['otile1','otile2','otile3','otile4']
            });
            map.addLayer(mapquestLayer);
        </script>
    </body>
</html>
```

Recorrer los datos en CSV y formatearlos como un objeto GeoJSON , el cual Leaflet puede fácilmente cargar.

```
<!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>Incendios en España desde 2001 hasta 2011</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>
```

Agregar una ventanita al mapa que muestre cuantos muertos por el incendio.

```
<!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>Incendios en España desde 2001 hasta 2011</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>
```

Agregar un enlace a la página de detalles del incendio.

```
<!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>Incendios en España desde 2001 hasta 2011</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>
```

Commitear el código generado.

```
$ git add .
$ git commit -m "Agregue un mapa a la página principal."
$ git push origin master
```

Abrir `detail.html` y agregar un mapa alli también, enfocado en el lugar del incendio.

```
<!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 "Agregue un mapa a la página de detalles."
$ git push origin master
```

CAPÍTULO 7

Act 5: Hola Internet

En este último capitulo, vamos a publicar tu aplicación en Internet usando Frozen Flask, una libreria de Python que guarda todas las páginas que hiciste con Flask como archivos estaticos que pueden ser cargados a la web fácilmente. Esto es una alternativa de publicación que no require un servidor de internet que tenga Python configurado..

Primero, usar pip para instalar Frozen Flask desde la linea de comandos.

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

Crear un nuevo archivo llamado `freeze.py` donde vamos a configurar que páginas deben ser convertidas a archivos estáticos.

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

Usar el editor de texto para escribir un archivo con la configuración basica de Frozen Flask.

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

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

Ejecutarlo desde la linea de comandos. Va a acrear una carpeta llamada `build` llena de los archivos estáticos generados.

```
$ python freeze.py
```

Ahora usar el navegador para abrir uno de los archivos locales creados en `build`, en vez de visitar las páginas generadas dinamicamente en `localhost`.

Si sigues mirando los archivos generados en la carpeta `build` puedes observar que sólo se generaron estaticos desde `index.html`, y no todas las páginas de detalles que nuestro template podria generar usando el archivo de datos.

Para generar estáticos de esos, nuevamente edita `freeze.py` para darle instrucciones de como hacer una página de cada registro en la fuente 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()
```

Ejecutalo nuevamente desde la linea de comandos y observa todas las otras páginas generadas en el directorio `build`. Intenta abrir una con tu navegador.

```
$ python freeze.py
```

Commitear todas las páginas estaticos al repositorio.

```
$ git add .
$ git commit -m "Estaticos de mi app"
$ git push origin master
```

Finalmente, publicaremos estos archivos estáticos a la web usando GitHub's Pages. Lo único que necesitamos es crear un nuevo branch en nuestro repositorio llamado `gh-pages` y publicar nuestros archivos alli en GitHub. Recuerda que hay muchas otras opciones para publicar archivos estáticos, desde Dropbox a Amazon's S3 service.

```
$ git checkout -b gh-pages # Crea una nueva branch
$ git merge master # Levantar todo el código desde el branch master
$ git push origin gh-pages # Publicarlo en GitHub desde tu nueva branch
```

En un o dos minutos, puedes visitar `http://<yourusername>.github.io/first-news-app/build/index.html` para ver tu aplicación publicada en la web.