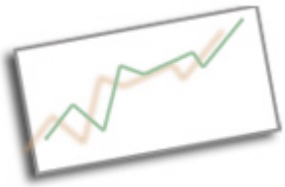


Coding and Data Skills
Dr. Cindy Royal
Texas State University
School of Journalism and Mass Communication

News Application

We've learned a lot of skills that could lend toward making a news application. You can use JavaScript to go through a JSON file and present data based on a user's input. You've learned how to use JavaScript-based charts that you can change, also based on a user's input. This tutorial will show you a more comprehensive approach to making a news application, one in which you have the ability to look at a full dataset and create pages to represent each line of data. It incorporates mapping features. This approaches the ProPublica model of the far-near view of data.

We'll be following this tutorial created by Ben Welsh of the LA Times. http://first-news-app.readthedocs.org/ . Ben did this tutorial at NICAR. He used data from the Los Angeles area, deaths during the LA Riots in 1992. We will use a different dataset.

To do this tutorial, you need:

- the Terminal
- a text editor like Text Wrangler
- Python
- the pip package manager
- the virtual environment

We will use Leaflet.js to make the maps. We will check to be sure that each program/package is installed, working on the masscomm login (so we have permissions). We'll post everything to GitHub at the end. You can find everything in my GitHub, github.com/cindyroyal/news-app-files.

Finding the Data

I found a more local dataset on data.world, one that has a listing of area parks. I worked with the data to remove any problems and to isolate the columns I wanted to use. I had to use the Text to Columns feature in Excel to get the long and lat in different columns. I found the data at https://data.world/cityofaustin/99qw-4ixs. There are 279 parks in the remaining dataset. I even started finding parks images for a few parks, so you can see how it works to include an image in your html. Finally, I added a column with a sequential id by starting the first two rows (1,2), then using the button at the bottom of the cell to continue down the column.

I then saved it as austinparks.csv from Excel, so I could use it in the Tutorial in the same way as in the First News App tutorial.

	A	B	C	D	E	F	G	H	I	J	K	L
1	GEODB_OID	long	lat	PARK_ID	PARK_NAME	ADDRESS	ZIP_CODE	PARK_ACRES	YEAR_OPEN	PARK_STATU	PARK_TYPE	img
2	1	-97.75883915	30.25545673	312	West Bouldin Creek Greenbelt	1200 S 6th St., Austin, Texas 78704	78704	16.8673136		Open	Greenbelt	bouldingreenb
3	2	-97.78953227	30.42900167	214	Mountain View Neighborhood Park	9000 Middlebie Rd., Austin, Texas 78750	78750	8.27796311	1997	Open	Neighborhood	mountainview
4	3	-97.67498937	30.27766276	217	Norman School Park	3901 Tannehill Ln., Austin, Texas 78721	78721	7.08904959	1975	Open	School	
5	4	-97.78069465	30.359748	218	North Cat Mountain Greenbelt	6801 Cat Creek Trl., Austin, Texas 78731	78731	30.7166135	1986	n_Restricted	Greenbelt	northcatmoun
6	5	-97.73599541	30.23398562	388	Heritage Oaks Neighborhood Park	2100 Parker Ln., Austin, Texas 78741	78741	3.53193169	2011	Open	Neighborhood	
7	6	-97.6999759	30.33286643	281	St. John's Pocket Park	889 Wilks Ave., Austin, Texas 78752	78752	0.86356904	1964	Open	Pocket	
8	7	-97.68422161	30.42470236	310	Wells Creek Greenbelt	13120 Metric Blvd., Austin, Texas 78727	78727	9.24365862	1991	n_Restricted	Greenbelt	
9	8	-97.74728495	30.37682359	178	Hill School Park	8405 Tallwood Dr., Austin, Texas 78759	78759	4.70107593	1975	Open	School	
10	9	-97.66304692	30.47292007	276	Springbrook Driving Range	1800 Picadilly Rd., Pflugerville, Texas 78664	78664	53.5789742	1988	Open_Fee	Special	
11	10	-97.75856024	30.29517274	361	Hartford Planting Strip	2516 Hartford Rd., Austin, Texas 78703	78703	1.72721253	1969	Closed	nting Strips/Triangles	
12	11	-97.74688768	30.25165564	360	The Circle ROW Greenbelt	1300 The Circle, Austin, Texas 78704	78704	1.20806081	2008	Open	Greenbelt	
13	12	-97.73333429	30.31347795	365	Triangle Commons Neighborhood Park	722 W 46th St., Austin, Texas 78751	78751	6.0218528	2005	Open	Neighborhood	
14	13	-97.78522838	30.23447446	272	South Austin Senior Activity Center	3911 Manchaca Rd., Austin, Texas 78704	78704	4.52988616		Open	Special	
15	14	-97.7806245	30.43061525	288	Tanglewood Neighborhood Park	11409 Rustic Rock Rd., Austin, Texas 78750	78750	14.1204649		Open	Neighborhood	
16	15	-97.68965268	30.2779001	396	Plummers Cemetery	1150 Blk of Springdale Rd., Austin, Texas 78721	78721	7.12815852		Open_Restricted	Cemetery	
17	16	-97.65102145	30.3947847	398	Oertli Neighborhood Park	12613 Blaine Rd., Austin, Texas 78753	78753	6.13351119	2013	Planned	Neighborhood	
18	17	-97.7421783	30.15544328	171	Grand Meadow Neighborhood Park	8022 Thaxton Dr., Austin, Texas 78747	78747	9.99401021	1988	Open	Neighborhood	
19	18	-97.74335364	30.24708331	194	Little Stacy Neighborhood Park	1500 Alameda Dr., Austin, Texas 78704	78704	6.86158307	1929	Open	Neighborhood	
20	19	-97.76921913	30.41753092	226	Oakview Neighborhood Park	10902 Oak View Dr., Austin, Texas 78759	78759	6.97521809	1981	Open	Neighborhood	
21	20	-97.83260569	30.19351612	346	Piney Bend Neighborhood Park	8601 Piney Creek Bnd., Austin, Texas 78745	78745	4.20313079		Open	Neighborhood	
22	21	-97.72237551	30.27551688	395	Oakwood Annex Cemetery	1509 E MLK Jr Blvd., Austin, Texas 78702	78702	18.576733		Open_Restricted	Cemetery	
23	22	-97.6990881	30.2376668	260	Roy G. Guerrero Colorado River Metro Park	400 Grove Blvd., Austin, Texas 78741	78741	399.466583		Open	Metropolitan	
24	23	-97.78771016	30.29201345	251	Red Bud Isle	3401 Red Bud Trl., Austin, Texas 78746	78746	17.493079		Open	Special	
25	24	-97.74898954	30.26450549	299	Shoal Beach at Town Lake Metro Park	707 W Cesar Chavez St., Austin, Texas 78701	78701	15.1151787		Open	Metropolitan	
26	25	-97.78098508	30.44956971	387	Springwoods Neighborhood Park	9117 Anderson Mill Rd., Austin, Texas 78729	78729	12.4514559		Open	Neighborhood	
27	26	-97.74789801	30.26754902	255	Republic Square	422 Guadalupe St., Austin, Texas 78701	78701	1.74910215		Open	Special	
28	27	-97.78765933	30.40467121	417	Maggie Boatright Area of the Bull Creek Greenbelt	958 Spicewood Springs Rd., Austin, Texas 78759	78759	8.99115991	1996	Open	Greenbelt	
29	28	-97.74882059	30.27717578	381	Heath Eiland and Morgan Moss BMX Skate Park	1213 Shoal Creek Blvd., Austin, Texas 78701	78701	1.55637564	1931	Open	Special	
30	30	-97.75642352	30.1965727	179	Houston School Park	5506 Tallow Tree Dr., Austin, Texas 78744	78744	8.56149739	1977	Open	School	
31	31	-97.71381696	30.24913273	297	Longhorn Shores at Town Lake Metro Park	60 S Pleasant Valley Rd., Austin, Texas 78741	78741	10.6360616		Open	Metropolitan	
32	32	-97.76641625	30.24189455	273	South Austin Tennis Center	1008 Cumberland Rd., Austin, Texas 78704	78704	12.0210417	1980	Open	Special	
33	33	-97.76851196	30.24093604	270	South Austin Neighborhood Park	1100 Cumberland Rd., Austin, Texas 78704	78704	11.7878397	1964	Open	Neighborhood	
34	34	-97.68869004	30.29886553	401	Tannehill Creek Greenbelt	2403 E 51st St., Austin, Texas 78723	78723	4.58490029	2012	Planned	Greenbelt	

Starting our News App

So with our edited csv of the parks, we are ready to make an application. We are going to create an application that lists all parks from the csv, maps them and then creates a separate page for each park. Since there are 279 on the list, it would take a really long time to do each of these individually. So, we are going to use the Flask framework to help us. Flask is a Python micro-framework http://flask.pocoo.org/. It differs from a framework like Ruby on Rails in that it is more lightweight, simpler and doesn't provide a database for you. But that's fine for this project. We are going to use our csv for the data. We are also going to ultimately "freeze" our site to make a static site – generating all the html pages, so we don't have to host the site on a Python-enabled server. This will make sense as we go.

The documentation for the First News App explains what you need and how to get it. I have already installed Python, PIP and virtualenv on our computers. PIP lets you install things and the virtualenv sets up an environment that lets us run all the required things we need, basically hosting a lightweight server on our computer while we create the app.

You can do these commands in the Terminal, to check if you have each. Remember that the $ just represents the prompt, so you don't type it.

```
$ python --version
$ pip --version
$ virtualenv --version
```

If you don't have something, please reference the First News App tutorial to see how to download each item.

Creating our Environment
Start by creating the Virtual Environment. You can navigate in Terminal to anywhere you want to start this app. I am just going to use the root of the user. If you ever need to get to that root in the Finder, click Cmd-Shift-H and it will bring you there. A virtual environment allows you to set up everything you need to work with your program, so you can be working in different versions for different projects.

```
$ virtualenv parks-app
```

cd to that folder and turn on the virtual environment with the activate command below. You will need to activate the project each time you need to work on it (you don't need to create a new virtualenv each time). Notice how the command line looks once you activate. It includes the virtualenv name in parentheses at the beginning of the prompt.

```
$ cd parks-app
$ . bin/activate
```

Ben's tutorial uses the touch command to create new files. You can do that, but you can also just create a new file in Text Wrangler when it is necessary. We will be creating two python scripts - app.py and later freeze.py. And we'll have two html templates - index.html and detail.html.

Installing Flask
For your application, you will need to install Flask.

```
$ pip install Flask
```

Go to Text Wrangler and create a new file. Save it in the parks-app directory as **app.py**. It is VERY important that you save things in the right place. This is the nature of using a framework.

Put this code in the **app.py** file and save it. This file will do all the routing for us in the application. We'll be adding to it as we go. We are importing Flask, rendering a template and creating an index page. We are also including the app/route to connect to the root of our site.

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

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

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Now create a templates folder. You can do this through the finder or the Terminal.

$ mkdir templates

Create a file called **index.html** in the templates folder and put in some text like: Hello World. Make sure to save it. We are just testing to see if it works.

Save and go to the Terminal, run app.py. When we are running the app, we are simulating a python server environment that will allow the framework functions to execute.

$ python app.py

You'll see some code to indicate the app is running. Go to a browser and open:

localhost:5000

Localhost is the root of your server. This only works when you have started the virtual environment.

Adding the Data
You should see "Hello World" come up in your index page. Congrats. Your app is on its way.

Make a directory called static and include your csv in there. The static directory should be below the parks-app directly. Do this in the Finder, so you can also move over your csv. Remember, if you need to get to the user directory on a Mac, use Cmd-Shift-H

Now make the following changes to **app.py**. This imports the csv module. The def get_csv() function allows you to open a csv and create objects from the items.

Notice the line that has csv_file = open(csv_path, 'rU'). I added the U to take care of Unicode and newline characters in the data.

Also notice the reference to austinparks.csv. You would change this if you used a different csv.

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

def get_csv():
    csv_path = './static/austinparks.csv'
    csv_file = open(csv_path, 'rU')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)
    return csv_list

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

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Save it.

Then go to the index.html file and remove “Hello World”.

Add this code to the file. This will check to see if any of the data is working.

```
<!DOCTYPE html>
<html lang="en">
    <head></head>
    <body>
        <h1>Austin Parks</h1>
        {{ object_list }}
    </body>
</html>
```

The code in the curly braces indicates a call to the csv. Reference that in def index() function in **app.py**.

If the app is still running, go to the browser at localhost:5000 and refresh. If it is not still running, run the command to start the app again ($ python app.py). You should see the data all jumbled on the screen.

MVC Architecture
Just a note about how Model-View-Controller or MVC architecture works. Most data-based frameworks separate these responsibilities, so that the model speaks to the database and connects to controllers that drive the application. This information is passed to the View which is rendered through the html templates. Learn more at https://realpython.com/blog/python/the-model-view-controller-mvc-paradigm-summarized-with-legos/

Formatting the Data on the Page
Now we will work to format it the data better. Change index.html:

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

    </head>
    <body>

        <h1>Austin Parks</h1>
        <table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
                <th>Address</th>
                <th>Acres</th>
            </tr>
            {% for obj in object_list %}
            <tr>
                <td><a href="{{ obj.id }}/">{{ obj.name
}}</a></td>
                <td>{{ obj.address }}</td>
                <td>{{ obj.acres }}</td>
            </tr>
            {% endfor %}
        </table>

    </body>
</html>
```

We are creating a table that includes all the items in the csv. The items in the double curly braces represent reading the elements from the csv data. Python/Flask provides the loop right in the html to go through the csv, writing each element as items in the table.

You should be able to reload the localhost:5000 page and see the complete table!

Notice that the id is used to create the url for the detail page for each park. We will create that next.

Austin Music Venues

Name	Address	Acres	Year Opened	Status	Type
West Bouldin Creek Greenbelt	1200 S 6th St., Austin, Texas 78704	16.86731362		Open	Greenbelt
Mountain View Neighborhood Park	9000 Middlebie Rd., Austin, Texas 78750	8.27796311	1997	Open	Neighborhood
Norman School Park	3901 Tannehill Ln., Austin, Texas 78721	7.08904959	1975	Open	School
North Cat Mountain Greenbelt	6801 Cat Creek Trl., Austin, Texas 78731	30.71661354	1986	Open_Restricted	Greenbelt
Heritage Oaks Neighborhood Park	2100 Parker Ln., Austin, Texas 78741	3.53193169	2011	Open	Neighborhood
St. John's Pocket Park	889 Wilks Ave., Austin, Texas 78752	0.86356904	1964	Open	Pocket
Wells Creek Greenbelt	13120 Metric Blvd., Austin, Texas 78727	9.24365862	1991	Open_Restricted	Greenbelt
Hill School Park	8405 Tallwood Dr., Austin, Texas 78759	4.70107593	1975	Open	School
Springbrook Driving Range	1800 Picadilly Rd., Pflugerville, Texas 78664	53.57897418	1988	Open_Fee	Special
Hartford Planting Strip	2516 Hartford Rd., Austin, Texas 78703	1.72721253	1969	Closed	Planting Strips/Triangles
The Circle ROW Greenbelt	1300 The Circle, Austin, Texas 78704	1.20806081	2008	Open	Greenbelt
Triangle Commons Neighborhood Park	722 W 46th St., Austin, Texas 78751	6.0218528	2005	Open	Neighborhood
South Austin Senior Activity Center	3911 Manchaca Rd., Austin, Texas 78704	4.52988616		Open	Special
Tanglewood Neighborhood Park	11409 Rustic Rock Rd., Austin, Texas 78750	14.12046488		Open	Neighborhood
Plummers Cemetery	1150 Blk of Springdale Rd., Austin, Texas 78721	7.12815852		Open_Restricted	Cemetery
Oertli Neighborhood Park	12613 Blaine Rd., Austin, Texas 78753	6.13351119	2013	Planned	Neighborhood
Grand Meadow Neighborhood Park	8022 Thaxton Dr., Austin, Texas 78747	9.99401021	1988	Open	Neighborhood
Little Stacy Neighborhood Park	1500 Alameda Dr., Austin, Texas 78704	6.86158307	1929	Open	Neighborhood
Oakview Neighborhood Park	10902 Oak View Dr., Austin, Texas 78759	6.97521809	1981	Open	Neighborhood

Adding the Detail Pages

Next we want those links in the name column to work. They will go to a special page for each park.

In **app.py**, include this function for the detail section. Do this below the index function but before the final if statement. Pay attention to spacing.

```
@app.route('/<row_id>/')
def detail(row_id):
    template = 'detail.html'
    object_list = get_csv()
    for row in object_list:
        if row['id'] == row_id:
            return render_template(template, object=row)
```

We are creating a row id to correspond to the url, looping through the csv to match the url and passing that matching row to the detail template.

Create a file in your templates directory called **detail.html** and include this in it.

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

    </head>
    <body>
        <h1>{{ object.name }}</h1>
        <img src=
        <p>Address: {{ object.address }}</p>
        <p>Acres: {{ object.acres }}</p>
        <p>Year Opened: {{ object.opened }}</p>
        <p>Status: {{ object.status }}</p>
        <p>Type: {{ object.type }}</p>


    </body>
</html>
```

Save everything. Restart the server (ctrl-C if it is still running) with **app.py**. Go back to localhost:5000 and refresh. You should see the full table there and be able to click on one of the names to get to its page. The url for the details pages is localhost:5000/1/ … etc. You should see detail pages that look something like this.

North Cat Mountain Greenbelt

Address: 6801 Cat Creek Trl., Austin, Texas 78731

Acres: 30.71661354

Year Opened: 1986

Status: Open_Restricted

Type: Greenbelt

Of course, you can, and should add better styling and layout for these pages. This tutorial just goes through the functionality. You can add any html you need in index.html and detail.html, and you should include a stylesheet.

It's probably a good idea to include links to a stylesheet now, because you can use the templating features to include it on each detail page. Put the code below in **detail.html**. By putting the css folder in the "static" folder, the app will be able to find it and include it in the build.

```
<link rel="stylesheet" href="../static/css/style.css"
type="text/css" />
```

This finds the css folder in the static folder above the details pages. This will make sense when we make the static site.

Put this code in the head of the **index.html** page. This finds the css folder below the parks-app folder.

```
<link rel="stylesheet" href="static/css/style.css"
type="text/css" />
```

You can add the css folder and style.css in the static folder either before or after you create the static site. Your files are all hooked up!

Adding Images
Some of our parks have images. And we might want to get images from all parks at some point. I have added names of images in the csv. For those without images, I created a placeholder image called no.jpg, and included that in the csv in the img column. The images need to exist somewhere. Create an "img" folder under the "static" folder and place your images in there. As with the css folder, this will be retained in the build below.

You can add a line in **detail.html** in the html section where you'd like to include the image. See how the script uses both html and Flask to reference the image from the data. Add this code below the name of the park.

```
<img src="../static/img/{{ object.img }}" height = "200px" />
```

Adding Web Addresses
Let's say you have web addresses in your csv. I found a few for some parks, and then just used the Austin Parks Foundation search page address for the rest. You would want to find urls for all (or most) of the parks (or for whatever data you are using). Include this line in your **detail.html** page to replace h1 holding the name. This adds the url to the name.

```
<h1><a href="{{ object.web }}" target="_blank">{{
object.name }}</a></h1>
```

Error Handling
Ben's tutorial also includes the ability to handle errors in the url. The entire app.py should look like this now. See the abort import and statement in the detail function.

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

def get_csv():
    csv_path = './static/austinparks.csv'
    csv_file = open(csv_path, 'rU')
    csv_obj = csv.DictReader(csv_file)
    csv_list = list(csv_obj)
    return csv_list

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

@app.route('/<row_id>/')
def detail(row_id):
    template = 'detail.html'
    object_list = get_csv()
    for row in object_list:
        if row['id'] == row_id:
            return render_template(template, object=row)
    abort(404)

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

Try choosing another number for a detail page that doesn't exist, like localhost:5000/9999/. It gives a "File Not Found" message instead of breaking the app.

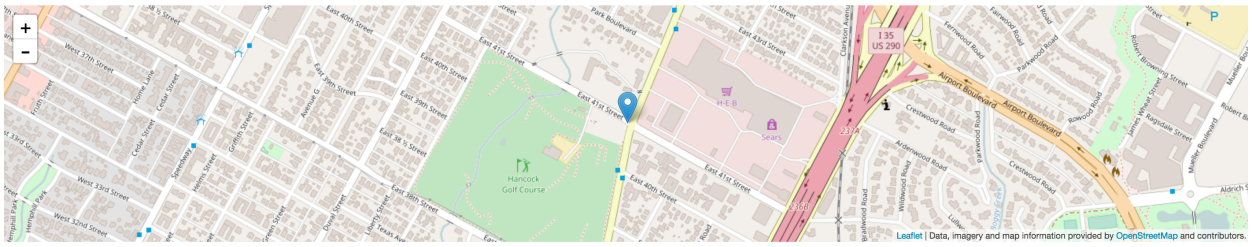
Adding the Maps
This is a great app. But it could be even better if we could map those items. We are going to create a map for the index page that maps the location for all the parks and then individual maps for each park page. Bet you're glad you aren't coding each one.

For this, we are going to include a nice JavaScript library for mapping called Leaflet.js.

Adding the Map to the Detail Page
To add the map to the detail page, make the following changes in **detail.html.** It includes the leaflet libraries in the head, the place where the map is drawn in the page (div) and the script to create it. The script relies on the x and y coordinates (longitude and latitude) in the data.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css" />
    <script
src="https://unpkg.com/leaflet@1.0.3/dist/leaflet.js"></script>
    </head>
    <body>
    <div id="map" style="width:100%; height:300px;"></div>
        <h1>{{ object.name }}</h1>
        <p>Address: {{ object.address }}</p>
        <p>Acres: {{ object.acres }}</p>
        <p>Year Opened: {{ object.opened }}</p>
        <p>Status: {{ object.status }}</p>
        <p>Type: {{ object.type }}</p>

<script type="text/javascript">
            var map = L.map('map').setView([{{ object.y }}, {{
object.x }}], 16);
            var osmLayer = new
L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
{
                maxZoom: 18,
                attribution: 'Data, imagery and map information
provided by <a href="http://www.openstreetmap.org/"
target="_blank">OpenStreetMap</a> and contributors.'
            });
            map.addLayer(osmLayer);
            var marker = L.marker([{{ object.y }}, {{ object.x
}}]).addTo(map);
</script>

    </body>
</html>
```

Reload one of the detail pages to see the maps. You should be able to see a map on every detail page with a pin for the location.



Hancock Golf Course

Address: 811 E 41st St., Austin, Texas 78751

Acres: 50.39607807

Year Opened: 1946

Status: Open_Fee

Type: Golf Course

Adding a Map to the Main Page

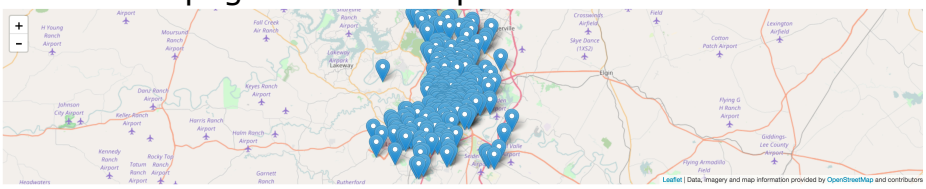
Modify **index.html** to look like this. It includes the leaflet libraries in the head, the place where the map is drawn in the page (div) and the script to create it. The script relies on the x and y coordinates in the data and it even includes the name and link to the individual page in the pin popup. I have adjusted the set view for Austin's lat and long and an initial zoom level, but the user can zoom and navigate the map on the site. You would adjust these depending on the location of your app.

```
<!DOCTYPE html>
<html lang="en">
    <head>
   <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.0.3/dist/leaflet.css" />
        <script
src="https://unpkg.com/leaflet@1.0.3/dist/leaflet.js"></script>
    </head>

<body>
        <div id="map" style="width:100%; height:300px;"></div>
        <h1>Austin Parks</h1>
        <<table border=1 cellpadding=7>
            <tr>
                <th>Name</th>
                <th>Address</th>
                <th>Acres</th>
            </tr>
            {% for obj in object_list %}
            <tr>
```

```
                <td><a href="{{ obj.id }}/">{{ obj.name
}}</a></td>
                <td>{{ obj.address }}</td>
                <td>{{ obj.acres }}</td>
            </tr>
            {% endfor %}
        </table>

    <script type="text/javascript">
          var map = L.map('map').setView([30.3, -97.7], 10);
          var osmLayer = new
L.TileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
{
              maxZoom: 18,
              attribution: 'Data, imagery and map information
provided by <a href="http://www.openstreetmap.org/"
target="_blank">OpenStreetMap</a> and contributors.'
          });
          map.addLayer(osmLayer);
          var data = {
            "type": "FeatureCollection",
            "features": [
              {% for obj in object_list %}
              {
                "type": "Feature",
                "properties": {
                  "name": "{{ obj.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.name +
                        '</a>'
                    );
                }
            });
```

```
            map.addLayer(dataLayer);
</script>
</body>
</html>
```

Reload the page and click a pin.



Austin Parks

Name	Address
West Bouldin Creek Greenbelt	1200 S 6th St., Austin, Texas 78704
Mountain View Neighborhood Park	9000 Middlebie Rd., Austin, Texas 78750
Norman School Park	3901 Tannehill Ln., Austin, Texas 78721
North Cat Mountain Greenbelt	6801 Cat Creek Trl., Austin, Texas 78731
Heritage Oaks Neighborhood Park	2100 Parker Ln., Austin, Texas 78741
St. John's Pocket Park	889 Wilks Ave., Austin, Texas 78752
Wells Creek Greenbelt	13120 Metric Blvd., Austin, Texas 78727
Hill School Park	8405 Tallwood Dr., Austin, Texas 78759
Springbrook Driving Range	1800 Picadilly Rd., Pflugerville, Texas 78664
Hartford Planting Strip	2516 Hartford Rd., Austin, Texas 78703

How About a Search Field for the Main Page?

Let's use a little JavaScript to get a search going in the search page. We can use JQuery to provide a nice search capability. Do this before you build this site, so the code will be synchronized for any future builds.

Include the following in **index.html** below the h2 and above the table. And give your table an id="table1". **Don't forget to include the link to the JQuery library in the <head>.**

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.m
in.js"></script>
```

The code includes a form with a text input, some DOM elements to hold the search term and the number that result in the search. Then the script goes gets the value of the search term and converts to lowercase, looks at the value in the first column -- eq(0) -- converted to lowercase, then goes through the table, matches the search term to the first column and either shows or hides the row.

```
<form id="form1">
<p>Search: <input type="text" id="term" /></p>
<input type="submit" value="submit" />
</form>
<p>Search Term: <span id="answer"></span></p>
<p>Number: <span id="answer2"></span></p>

<script>
$(document).ready(function() {
$("#form1").submit(function() {
  j=0;
  $('#table1 tr').hide();
    //convert search term to lowercase
    var searchterm = $("#term").val().toLowerCase();
    $("#answer").html(searchterm);

    //for each row of table...
    $('#table1 tr').each(function() {
    //find the text in the 1st td, convert to lowercase
    var col_search =
$(this).find('td').eq(0).text().toLowerCase();

      // if input name is equal to the column name text, then
show it
      if (col_search.includes(searchterm)) {
      $(this).show();
      j=j+1;
      }// end if
       else {
      $(this).hide();
      }

      });// end for
     $("#answer2").html(j);

  return false;
});// end submit

});// end ready
</script>
```

Creating the Static Site

This site is awesome, but we want to host it on the Web. We could try to find a host that could support a Python/Flask environment. We need to do that if we are creating a site that users actively contribute to. But this site is static. We might update parks occasionally, but we'd update the csv and then rerun the app to do that. For this app, we can generate a static site that creates all

our pages, and then we can host it on any server (like our Reclaim or Bluehost accounts) that supports HTML, CSS and JavaScript.

To do this, we will use Frozen Flask, a library that saves every page in your app as a flat file. In the Terminal (use Ctrl-C to exit your app, if necessary), run:

$ pip install Frozen-Flask
(you might need to sudo this if you get a permission error)

Create a new file called **freeze.py** in parks-app directory. Put this code in it.

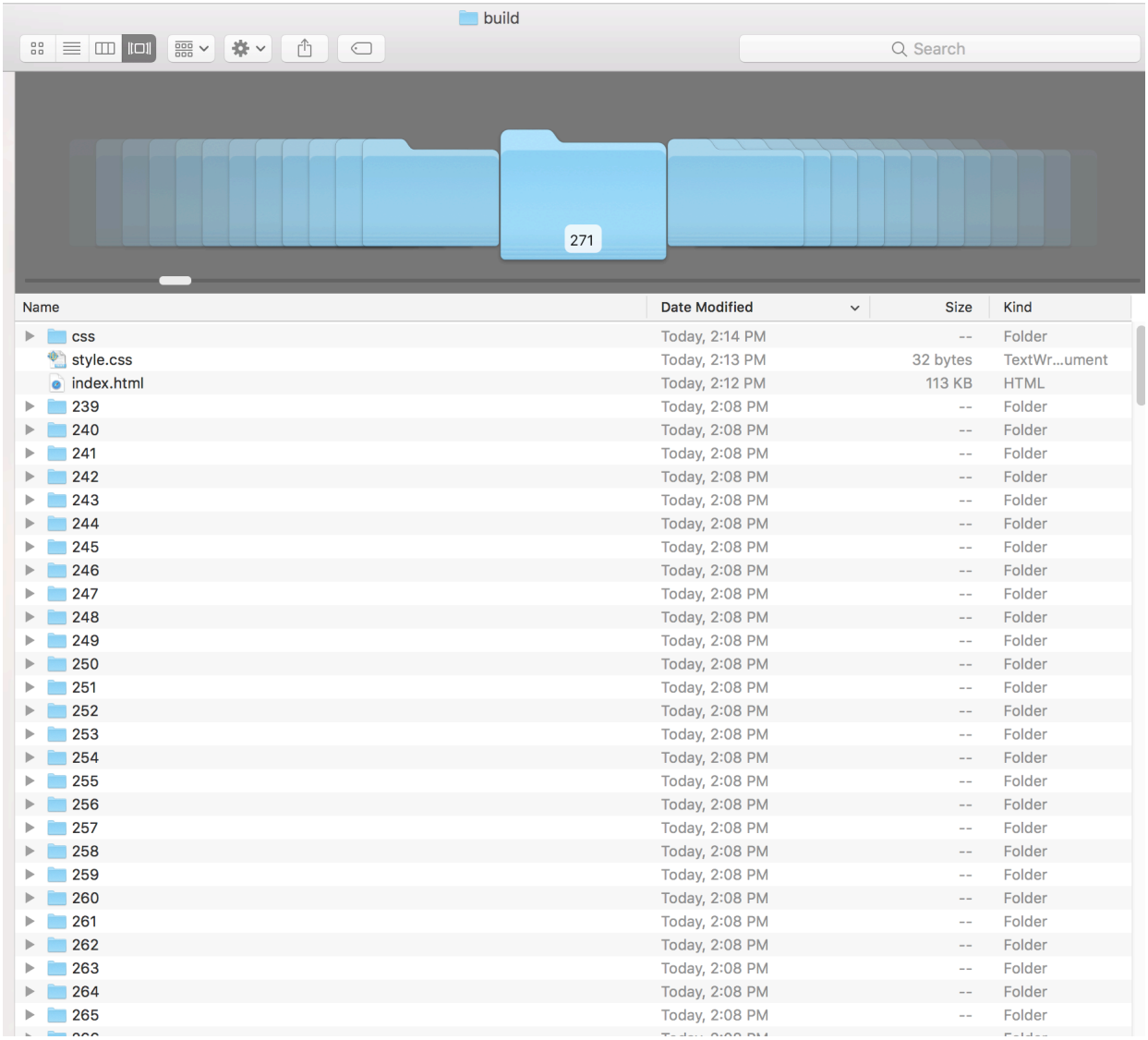
```
from flask_frozen import Freezer
from app import app, get_csv
freezer = Freezer(app)

@freezer.register_generator
def detail():
    for row in get_csv():
        yield {'row_id': row['id']}

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

Run freeze.py on the command line.

$ python freeze.py

This creates a build folder in which all your static files are located. Look for the build folder in your application in the Finder. The folder has index.html and folders with an index.html for each venue. It also includes the static folder with the csv in it. You can fetch these folders/files to a folder on your website to host the application there!

When you run the static site locally, you will see a directory for the detail pages. This will load the page properly when you host it on your Web server.



That’s it! You have created a beautiful, interactive application for 279 parks around Austin.

Of course, you should now go and commit this entire folder to a new GitHub repository, so you can share it with the world! Create a new repository and follow the instructions to push your entire parks-app directory to it.

Now you can see how all the different things we covered this semester come together. Think about how you could use this process for other types of

data, other types of storytelling. Completed files at various stages of this process can be found at github.com/cindyroyal/parks-app.

Now You Try

There's another dataset in the github repository. It's music-venues.csv.This is an abbreviated list of Austin music venues, with 19 venues in it. It has columns for name, address, phone, website, twitter, lat and long. And there are images in a folder for each, with names also in a column.

Go through the exercise again, using this dataset. The index.html page should include the name, address and phone for all, with the name and a link on the map. The detail pages should have name with linked url to their website, address, phone, and Twitter with the @name as the text and the url going to their Twitter page. Include the image below the name of the venue.

There are some things you may need to change. There are different variables in this dataset.

Get it to work in Flask, then save out as a static site with freeze.py.

If you have time, add the search capabilities before you run the build.

Give it a try!