Project 3: Weather Data Alerts

Celena Blackwood, Toni Edwards, Jacob Cortez, Julian Louden, Paola Mateo

REQUIREMENTS FOR THE REPORT

Part 1: Python Flask

Weather Alerts API was used, along with a JavaScript file that was displayed in a HTML/CSS format

Part 2: Web Scraping with Plotly

Web Scraping was done with a list of states to obtain their abbreviations, and Plotly was utilized to display a map along with the weather alerts that occur in that state

Part 3: Charts.js

For a JS library that we did not cover, we chose charts.js, to demonstrate the other 2 required visualizations

Part 4: Number of Records/User-Driven Interaction

Our data set was powered by over 100 records since we were able to receive alerts for all 50 states. Dropdowns in the index.html for all other websites were included, in order to easily access those diagrams.

DATA SOURCES

National Weather Service-Active Alerts

- https://www.weather.gov/documentation/services-web-api#/default/alerts_active
- Some of the parameters that this API holds includes, but is not limited to:
  - status
  - message type
  - event
  - area
  - urgency
- In this project, we only filtered it to the following: city, severity, event, urgency, headlines, and coordinates

List of U.S States Abbreviations

- https://www23.statcan.gc.ca/imdb/p3VD.pl?Function=getVD&TVD=53971
- Web scraping was done in order to identify the state abbreviations and add them as options that the user can select from in order to view the weather alerts for the specific state

DISPLAYING INFORMATION FROM API

Flask

- Set up the Flask API app with route to endpoints : Home(index), Map, Charts

Web Scraping

- Set up the web scraping function to retrieve state codes table and populate states menu
- Set up the python API function to request json response containing weather information as geoJSON object

Navigation Bar

- Add the navigation bar and states menu to html templates for all three endpoints

geoJSON created and collection insert to MongoDB

```
features.append(Feature(geometry=geometry, properties=properties))
feature_collection = FeatureCollection(features)
with open("feature_collection_geoJson.geojson", 'w') as g:
    dump(feature_collection, g)
client = MongoClient("localhost", 27017)
db = client["weather_db"]
collection = db["collections"]
collection.insert_one(feature_collection)
client.close()
feature_response = json_util.dumps(feature_collection)
return feature_response
```

```
from asyncio import Event, events
from distutils.log import set_verbosity
from werkzeug.utils import secure_filename
import weather_data
import scrape_states
import pandas as pd
from flask import Flask, render_template, request, redirect, jsonify, make_response
app = Flask(__name__)

state_codes = scrape_states.scrape()
print("This is the state code list:", state_codes)



@app.route("/", methods = ['POST', 'GET'])
def index():
    #home page for info and navigation
    return render_template("index.html", states = state_codes)
```

Initiating Flask

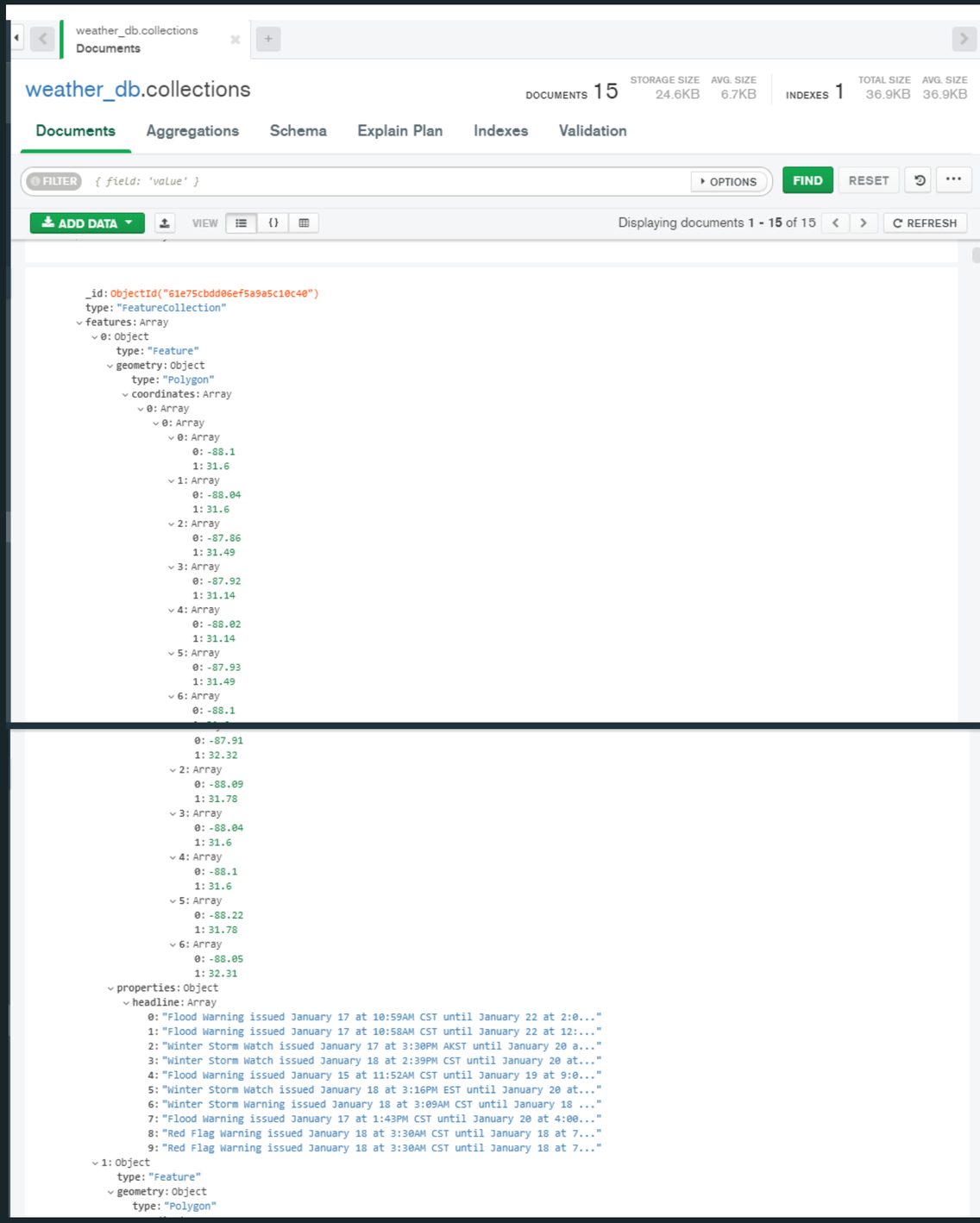
Web scraping done over state codes

Setting up routes

MONGODB

- MongoDB was used to import all of the data from the Weather API
- weather_db>collections
- From here, we are able to select the columns that we want to see and export them to the HTML files

mongodb://127.0.0.1:27017

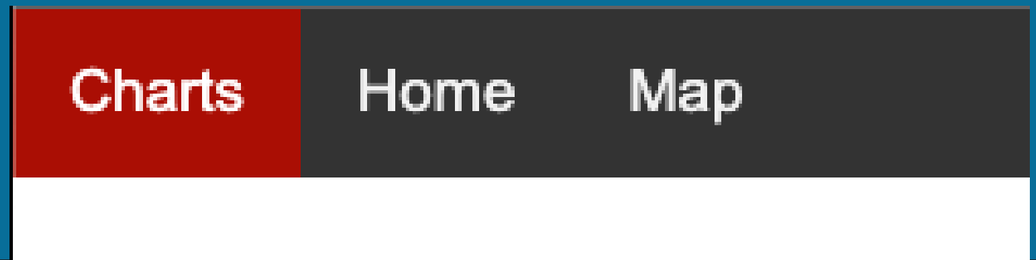


STYLE.CSS SHEET

```css
.topnav {
  overflow: hidden;
  background-color: #333;
}
.topnav a {
  float: left;
  color: #F2F2F2;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
  font-size: 17px;
}
.topnav a:hover {
  background-color: #ddd;
  color: black;
}
.topnav a.active {
  background-color: #AA0F04;
  color: white;
}
```

Features Added:

- Navigation bar to remain consistent for all three endpoints
- Red box to highlight which page user is currently in
- Utilizing the external style sheet allows us to make changes to multiple files, but only edit from 1 style sheet



INDEX.HTML

- Html code needs to make a call to flask, the endpoint
- Desired Components:
  - Navigation Bar
    - “Url_for” utilized for referencing back to additional web pages
  - Drop Down Menu for user to input desired state
    - Webscrapping file was created to obtain two letter state codes
  - Images
    - Rendering through Flask requirement again for usage of “url_for”
- Utilized CSS stylesheet and bootstrap for additional formatting
- Rendering of page dependent on organization of files used to compile it

{{url_for('/')}} function used to navigate through the other 2 websites

```
<div class="topnav">
  <a class="active" href="/">Home</a>
  <a href="{{url_for('map')}}">Map</a>
  <a href="{{url_for('charts')}}">Charts</a>
</div>
<br>
```

How user can select the state to see weather alerts:

```
<div class="menu">
<form id = "states_form" action="/data/${state}" methods="get">
  <select name="state" width = "300px">
    {% for state in states %}
    <option name={{state}}>{{state}}</option>
    {% endfor %}
  </select>
  <input type="submit" value="search">
</form>
</div>
```

MAP.HTML

Code to create the map

- Started out utilizing the weather function to grab all of the data needed across all 50 states
- Displaying information on leaflet and render the map on each individual page the user would select to get map alerts from.

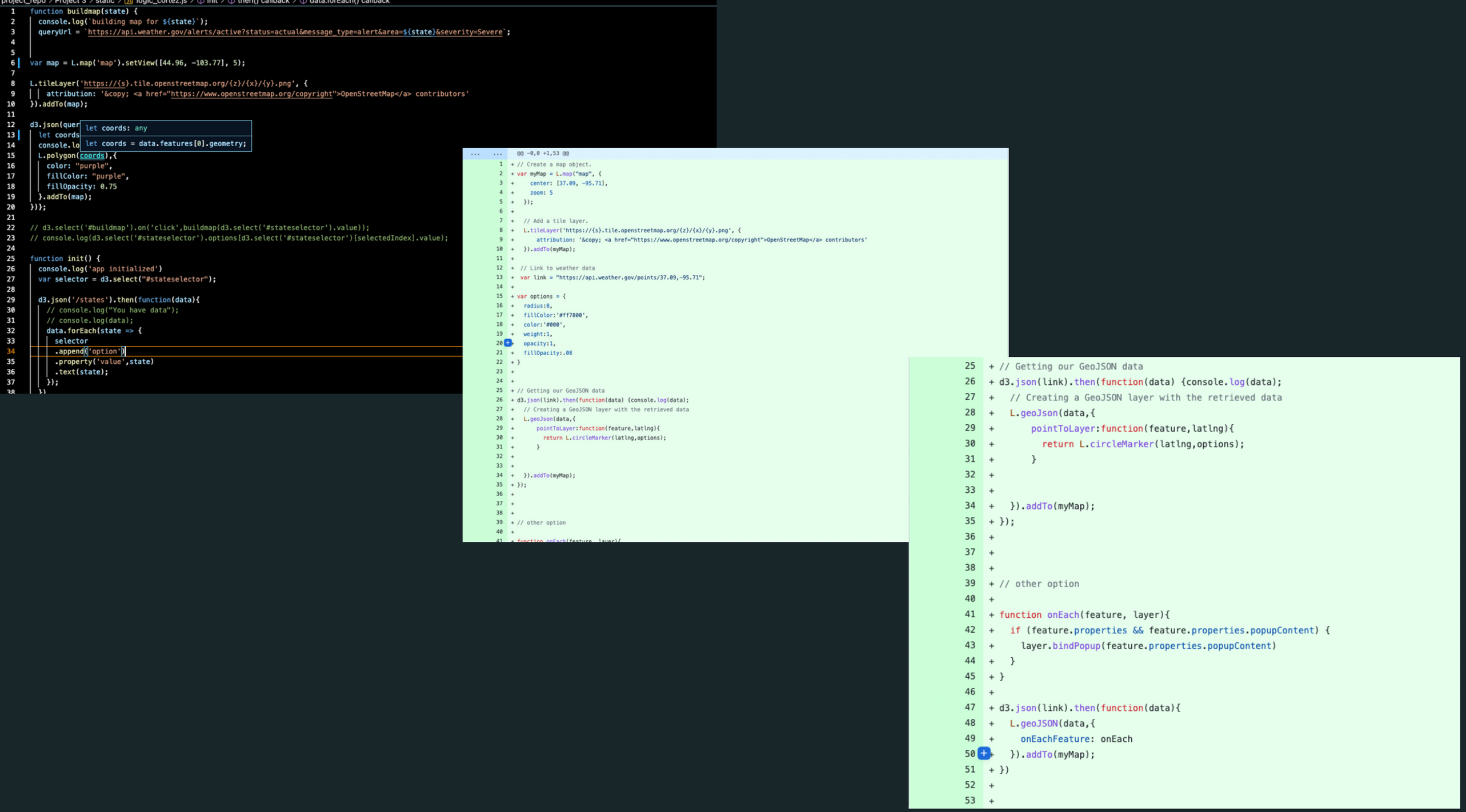
```
-- D3 Library -->
cript src="https://d3js.org/d3.v5.min.js"></script>

-- Leaflet CSS -->
ink rel="stylesheet" href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"
tegrity="sha512-xwE/Az9zrjBIphAcBb3F6JVqxf46+CDLwfLMHloNu6KEQCAWi6HcDUbeOfBIptF7tcCzusKFjFw2yuvEpDL9wQ=="
ossorigin="" />

-- Leaflet JavaScript code -->
cript src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js"
tegrity="sha512-gZwIG9x3wUXg2hdXF6+rVkLF/0Vi9U8D2Ntg4Ga5I5BZpVkVxlJWbSQtXPSiUTtC0TjtGOmxa1AJPuV0CPthew=="
ossorigin=""></script>

head>
ody>

iv class="states_menu">
orm>
<select id= "stateselector" onchange="buildmap(this.value)" name="state" width = "300px">
  <!-- {% for state in states %}
  <option value= {{state}}>{{state}}</option>
  {% endfor %} -->
</select>
<input id='buildmap' type="submit" value="search">
form>
div>
Call the API endpoint
-- Image -->
mg src="https://images.foxtv.com/static.fox26houston.com/www.fox26houston.com/content/uploads/2019/10/764/432/GFX_SEVERE_WEATHER_Large_Logo_PM_Blue_Red_HD1_Full_1460971142271_1193483
!-- The div where we'll insert our map -->
div id="map"></div>
-- JavaScript file -->
cript type="text/javascript" src='../static/logic_cortez.js'></script>
```


project_repo > Project 3 > static > logic_cortez.js > init > then() callback > data.forEach() callback
function buildmap(state) {
console.log(`building map for ${state}`);
queryUrl = `https://api.weather.gov/alerts/active?status=actual&message_type=alert&area=${state}&severity=Severe`;
var map = L.map('map').setView([44.96, -103.77], 5);
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
attribution: '© OpenStreetMap contributors'
}).addTo(map);
d3.json(quer
let coords: any
let coords = data.features[0].geometry;
let coords
console.lo
L.polygon(coords),{
color: "purple",
fillColor: "purple",
fillOpacity: 0.75
}.addTo(map);
});
// d3.select('#buildmap').on('click',buildmap(d3.select('#stateselector').value));
// console.log(d3.select('#stateselector').options[d3.select('#stateselector')[selectedIndex].value);
function init() {
console.log('app initialized')
var selector = d3.select("#stateselector");
d3.json('/states').then(function(data){
// console.log("You have data");
// console.log(data);
data.forEach(state => {
selector
.append('option')
.property('value',state)
.text(state);
});
@@ -0,0 +1,53 @@
+ // Create a map object.
+ var myMap = L.map("map", {
+ center: [37.09, -95.71],
+ zoom: 5
+ });
+ // Add a tile layer.
+ L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
+ attribution: '© OpenStreetMap contributors'
+ }).addTo(myMap);
+ // Link to weather data
+ var link = "https://api.weather.gov/points/37.09,-95.71";
+ var options = {
+ radius:8,
+ fillColor:'#ff7800',
+ color:'#000',
+ weight:1,
+ opacity:1,
+ fillOpacity:.08
+ }
+ // Getting our GeoJSON data
+ d3.json(link).then(function(data) {console.log(data);
+ // Creating a GeoJSON layer with the retrieved data
+ L.geoJson(data,{
+ pointToLayer:function(feature,latlng){
+ return L.circleMarker(latlng,options);
+ }
+ }).addTo(myMap);
+ });
+ // other option
+ function onEach(feature, layer){
+ // Getting our GeoJSON data
+ d3.json(link).then(function(data) {console.log(data);
+ // Creating a GeoJSON layer with the retrieved data
+ L.geoJson(data,{
+ pointToLayer:function(feature,latlng){
+ return L.circleMarker(latlng,options);
+ }
+ }).addTo(myMap);
+ });
+ // other option
+ function onEach(feature, layer){
+ if (feature.properties && feature.properties.popupContent) {
+ layer.bindPopup(feature.properties.popupContent)
+ }
+ }
+ d3.json(link).then(function(data){
+ L.geoJSON(data,{
+ onEachFeature: onEach
+ }).addTo(myMap);
+ })

```
@@ -0,0 +1,66 @@
+ // Store our API endpoint as queryUrl. This will be coming from app.py
+ var queryUrl = weather_response
+
+ // Perform a GET request to the query URL/
+ d3.json(queryUrl).then(function (weatherData) {
+   // Once we get a response, send the data.features object to the createFeatures function.
+   createFeatures(weatherData.features);
+ });
+
+ function createFeatures(mapData) {
+
+   // Define a function that we want to run once for each feature in the features array.
+   // Give each feature a popup that describes the place and time of the earthquake.
+   function onEachFeature(feature, layer) {
+     layer.bindPopup(`<h3>${feature.properties.place}</h3><hr><p>${new Date(feature.properties.time)}</p>`);
+   }
+
+   // Create a GeoJSON layer that contains the features array on the earthquakeData object.
+   // Run the onEachFeature function once for each piece of data in the array.
+   var earthquakes = L.geoJSON(earthquakeData, {
+     onEachFeature: onEachFeature
+   });
+
+   // Send our earthquakes layer to the createMap function/
+   createMap(earthquakes);
+ }
+
+ function createMap(earthquakes) {
+
+   // Create the base layers.
+   var street = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
+     attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
+   })
+
+   var topo = L.tileLayer('https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png', {
+     attribution: 'Map data: &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, <a href="http://viewfinderpanorama
+   });
+
+   // Create a baseMaps object.
+   var baseMaps = {
+     "Street Map": street,
+     "Topographic Map": topo
+   };
```

```
@@ -0,0 +1,30 @@
+ <!DOCTYPE html>
+ <html lang="en">
+ <head>
+   <meta charset="utf-8">
+   <title>Cortez Map *TEST*</title>
+
+   <!-- Leaflet CSS -->
+   <link rel="stylesheet" href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"
+     integrity="sha512-xwE/Az9zrjBIphAcBb3F6JVqxf46+CDLwfLMHloNu6KEQCAWi6HcDUbeOfBIptF7tcCzusKFjFw2yuvEpDL9wQ=="
+     crossorigin="" />
+
+   <!-- Leaflet JavaScript code -->
+   <script src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js"
+     integrity="sha512-gZwIG9x3wUXg2hdXF6+rVkLF/0Vi9U8D2Ntg4Ga5I5BZpVkVxlJWbSQtXPSiUTtC0TjtGOmxa1AJPuV0CPthew=="
+     crossorigin=""></script>
+
+   <!-- D3 library -->
+   <script src="https://d3js.org/d3.v5.min.js"></script>
+
+ </head>
+
+ <body>
+   <!-- map div -->
+   <div id="map"></div>
+
+   <!-- JavaScript file -->
+   <script type="text/javascript" src="../logic_cortez.js"></script>
+
+ </body>
+ </html>
```

```
@@ -0,0 +1,66 @@
+   // Create a baseMaps object.
+   var baseMaps = {
+     "Street Map": street,
+     "Topographic Map": topo
+   };
+
+   // Create an overlay object to hold our overlay.
+   var overlayMaps = {
+     Earthquakes: earthquakes
+   };
+
+   // Create our map, giving it the streetmap and earthquakes layers to display on load.
+   var myMap = L.map("map", {
+     center: [
+       37.09, -95.71
+     ],
+     zoom: 5,
+     layers: [street, earthquakes]
+   });
+
+   // Create a layer control.
+   // Pass it our baseMaps and overlayMaps.
+   // Add the layer control to the map.
+   L.control.layers(baseMaps, overlayMaps, {
+     collapsed: false
+   }).addTo(myMap);
+
+ }
```

```
@@ -1,31 +1,31 @@
- // Store our API endpoint as queryUrl. This will be coming from app.py
- var queryUrl = weather_response
+ // Store our API endpoint as queryUrl. This will be coming from app.py (weather_response )
+ var queryUrl = "https://api.weather.gov/alerts/urn:oid:2.49.0.1.840.0.f4d1b1c6788eb846c0d568f0d3c647573e1db0c9.001.1"

// Perform a GET request to the query URL/
d3.json(queryUrl).then(function (weatherData) {
  // Once we get a response, send the data.features object to the createFeatures function.
  createFeatures(weatherData.features);
});

- function createFeatures(mapData) {
+ function createFeatures(weatherData) {

  // Define a function that we want to run once for each feature in the features array.
  // Give each feature a popup that describes the place and time of the earthquake.
  function onEachFeature(feature, layer) {
-   layer.bindPopup(`<h3>${feature.properties.place}</h3><hr><p>${new Date(feature.properties.time)}</p>`);
+   layer.bindPopup(`<h3>${feature.properties.event}</h3><hr><p>${(feature.properties.areaDesc)}</p>`);
  }

  // Create a GeoJSON layer that contains the features array on the earthquakeData object.
  // Run the onEachFeature function once for each piece of data in the array.
- var earthquakes = L.geoJSON(earthquakeData, {
+ var weatheralearts = L.geoJSON(weatherData, {
    onEachFeature: onEachFeature
  });

  // Send our earthquakes layer to the createMap function/
- createMap(earthquakes);
+ createMap(weatheralearts);
}

- function createMap(earthquakes) {
+ function createMap(weatheralearts) {

  // Create the base layers.
  var street = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
```

```
@@ -44,7 +44,7 @@ function createMap(earthquakes) {

  // Create an overlay object to hold our overlay.
  var overlayMaps = {
-   Earthquakes: earthquakes
+   weatheralearts: weatheralearts
  };

  // Create our map, giving it the streetmap and earthquakes layers to display on load.
```

templates/map_cortez.html

```
@@ -24,7 +24,7 @@
    <div id="map"></div>

    <!-- JavaScript file -->
-   <script type="text/javascript" src="../logic_cortez.js"></script>
+   <script type="text/javascript" src="../logic_cortez2.js"></script>

  </body>
  </html>
```

CHARTS.HTML

- Library used=charts.js
- Charts.js is used to demonstrate data visualizations
  - For purposes of this project, only a bar graph and a pie chart were utilized
- The following package managers needed to be downloaded in order for JS library to successfully run:
  - Node
  - NPM

BAR GRAPH

- Used to analyze the events by severity
- Once state is selcted, we are able to view how many events there are and their "severity" levels

PIE CHART

- Used to view the type of events that occur by state
- User is able to hover over selections and notice the amount of a specific event that is occurring

```
with urlopen('https://api.weather.gov/alerts/active?status=actual&message_type=alert&area=TX&severity=Severe') as response:
    location = json.load(response)

location["features"]
```

✓ 0.9s

```
Output exceeds the size limit. Open the full output data in a text editor
[{'id': 'https://api.weather.gov/alerts/urn:oid:2.49.0.1.840.0.2a65aa07bcaeb1b92c717429e67fc713b4419ebc.001.1',
  'type': 'Feature',
  'geometry': None,
```

Once state is selected, only the columns that were being looked at were filtered:

```
# weather_df = pd.DataFrame({"city":city, "severity":severity,
# weather_df
weather_df=pd.DataFrame({"severity":severity, "event":event})
weather_df.value_counts()
```

✓ 0.7s

```
severity  event
Severe    Flood Warning           7
          Winter Storm Warning    5
          Winter Storm Watch      3
          Freeze Watch            2
          High Wind Warning       2
          Fire Weather Watch      1
dtype: int64
```

BAR GRAPH

```
<script>
  const ctx = document.getElementById('myChart').getContext('2d');
  const myChart = new Chart(ctx, {
      type: 'bar',
      data: {
        labels: ["Unknown", "Minor", "Moderate", "Severe", "Extreme"],
        datasets: [{
            data: [0,0,0,20,0],
            label: "Total",
            borderColor: "rgb(62,149,205)",
            backgroundColor: "rgb(62,149,205,0.1)",
            borderWidth:2
```

PIE CHART

```
const ctx1 = document.getElementById('myChart1').getContext('2d');
const myChart1 = new Chart(ctx1, {
    type: 'pie',
    data: {
      labels: ["Flood Warning","Winter Storm Warning", "Winter Storm Watch", "Freeze Watch", "High Wind Warning", "Fire Weather Watch"],
      datasets: [{
          data: [7,5,3,2,2,1],
          borderColor: [
            "#3cba9f",
            "#ffa500",
            "#c45850",
            "#76448a",
            "#21618c",
            "#f4d03f",
      ],
          backgroundColor: [
            "rgb(60,186,159,0.1)",
            "rgb(255,165,0,0.1)",
            "rgb(196,88,80,0.1)",
            "rgb(118,68,138,0.1)",
            "rgb(33,97,140,0.1)",
            "rgb(244,208,63,0.1)",
          ],
          borderWidth:2
        }
```

WEBSITES CREATED TO NAVIGATE THROUGH WEATHER ALERTS

- Index.html: http://127.0.0.1:5000/
- Map.html: http://127.0.0.1:5000/map
- Charts.html: http://127.0.0.1:5000/charts