|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
|  |  |  | |  |
| City of San Diego Get-it-Done Dashboard  Final Report for Project 2  UCSD Data Science and Visualization Bootcamp  April 2020 | | |
| Team Git it On:  Melissa Monroe | Architecture, design, data, Leaflet map, art  Andrew Bankston | Data, content, analysis, plots  Jacob Zacarias | Data, content, analysis, plots, maps  Annette Broeren | Powerpoint, final report, ReadMe | |  |

Table of Contents

[Articulation and Rationale 3](#_Toc69050834)

[Setting up 4](#_Toc69050835)

[Sources 4](#_Toc69050836)

[Inspiring Visualizations 5](#_Toc69050837)

[Original Sketches 6](#_Toc69050838)

[Execution 7](#_Toc69050839)

[Libraries 8](#_Toc69050840)

[New and mentionworthy: 8](#_Toc69050841)

[All libraries / dependencies 8](#_Toc69050842)

[Mongodb 9](#_Toc69050843)

[Dashboard 11](#_Toc69050844)

[Features 11](#_Toc69050845)

[Main Page 12](#_Toc69050846)

[Council District Page 13](#_Toc69050847)

[Conclusion 14](#_Toc69050848)

[Number of Service Requests by Year – Plotly 14](#_Toc69050849)

[Service Types 15](#_Toc69050850)

[Council Districts Performance 16](#_Toc69050851)

[Going forward 17](#_Toc69050852)

[code 18](#_Toc69050853)

[Flask Setup 18](#_Toc69050854)

[Main\_mbm.jpynb to process the data 20](#_Toc69050855)

[Visualization.py for the summary statistics 24](#_Toc69050856)

[App.py to populate main dashboard 24](#_Toc69050857)

[Cdapp.js to populate council districts dashboard 27](#_Toc69050858)

[Logic.JS to create map 28](#_Toc69050859)

[Charts 30](#_Toc69050860)

# Articulation and Rationale

The objective of this report is to describe the goal, process and observations of transforming a dataset into a dynamic dashboard for business metrics. For this project, City of San Diego’s Get-It-Done app data was used.

Get It Done San Diego is the official app for reporting non-emergency problems to the City of San Diego. App users can report problems like potholes or graffiti and connect directly to the City’s work tracking system.

The app has been in use since 2016 and the City of San Diego publishes annual Get-It-Done datasets in CSV format for historical data, and JSON format for the current year to date.

**Intended audience:** City of San Diego Leadership

**Decision:** Strategic/Operational – identify problem-areas, monitor and gage effectiveness of the Get-It-Done program overall and by Council district, using:

* Service request volume by period, type, council district
* Average response time from open to closed
* Open/closed deltas
* Service request volumes by type by council district

**Decision timing**: Ongoing for the life of the app

**Importance:** The dashboard can assist City Leadership in quickly identifying problem areas or problem periods, such as holidays, tourist season, etc.

**What actions can be taken**: The dashboard is a monitoring tool for City Leadership allowing for further investigation, remedying or use a pro-active approach as appropriate for the issue. For instance, during tourist season or while large conventions are in town, it is possible that mobile service requests for scooters spike. If the City had historical data to confirm this, the City could take a pro-active approach and request that the scooter service vendors increase their collection efforts during those times, thereby reducing the number of service requests, the amount of resouces the City would otherwise have to dedicate, and last but not least: reduce the nuisance factor for the public.

# Setting up

## Sources

City of San Diego Get It Done program files:   
<https://data.sandiego.gov/datasets/get-it-done-311/>

Used for Project:

Get It Done Requests JSON API – json

Get It Done Requests year-to-date – csv

Get It Done Requests 2020 – csv

Get It Done Requests 2019 – csv

Not used but listed for future consideration:

Get It Done Requests 2018 – csv

Get It Done Requests 2017 – csv

Get It Done Requests 2016 – csv9 CSV files

City Council Districts geojson:  
<https://data.sandiego.gov/datasets/city-council-districts/>

## Inspiring Visualizations

|  |  |
| --- | --- |
| Graphical user interface, application  Description automatically generated | Graphical user interface  Description automatically generated |
| Graphical user interface, chart  Description automatically generated | Graphical user interface, application  Description automatically generated |

## Original Sketches

Diagram

Description automatically generated

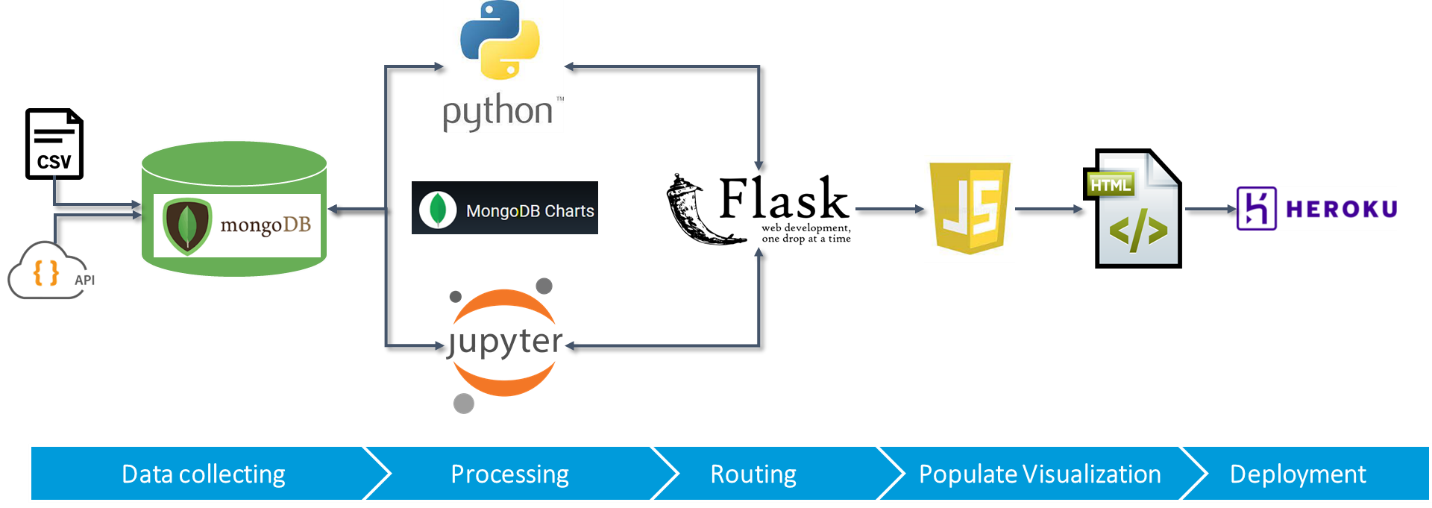
Graphical user interface, diagram

Description automatically generated

# Execution

The approach consisted of a two-tier environment where data from the CSVs is imported into a MongoDB database, and the year-to-date data published in JSON-format is called in via API.

* Download the source files to a Github repository with branches for all team members
* Create Atlas Cloud MongoDB
* Create documents from the CSV files in MongoDB converting strings to numbers or date values where appropriate
* Connect to mongodb, read collections, add group by and count service name, filter by date
* Create graphs in MongoDB Charts
* Create pre-populated summary and chart documents in MongoDB to reduce dashboard loading time
* Use NiceAdmin dashboard template and modify for purpose
* Use Python, Jupyter, Flask, Javascript, HTML/CCS to process and load data
* Populate and deploy dashboard



## Libraries

### New and mention-worthy:

**DNS E**nables connection to Atlas MongoDB.

**Flask CORS** cross origin resource sharing (enables running API calls when origin is different from web origin).

**Pytz** Accurate cross-platform time calculation.

**Django** stores datetime information in UTC, uses time-zone-aware datetime objects internally, and translates them to the end user’s time zone.

### All libraries / dependencies

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

## Mongodb

Approximately 680,000 service requests for 2019 and 2020 were imported into MongoDB. Strings were converted to numbers or dates where appropriate.

Table

Description automatically generated

Graphical user interface, text, application

Description automatically generated

# Dashboard

## Features

* Year selector
* Summary Status at a glance
* Service requests by service type
* Leaflet mapbox with popup tool tip and filter (service type, limit)
* Service request count by month
* Top 20 service requests by type
* Service requests by council district
* Average case length by council district
* COVID-19 heatmap

## Main Page

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

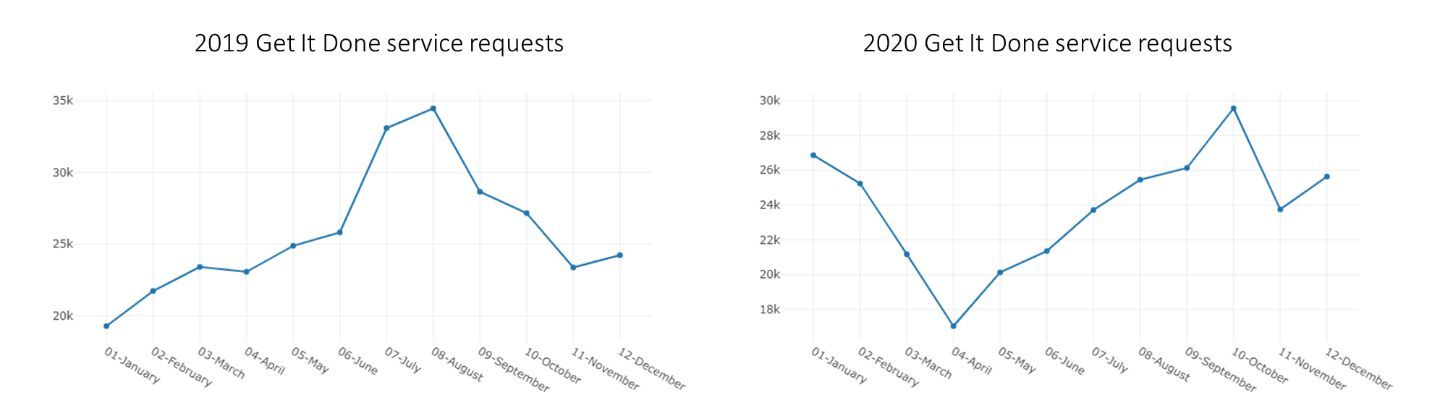
## Council District Page

Graphical user interface, chart

Description automatically generated

# Conclusion

## Number of Service Requests by Year – Plotly



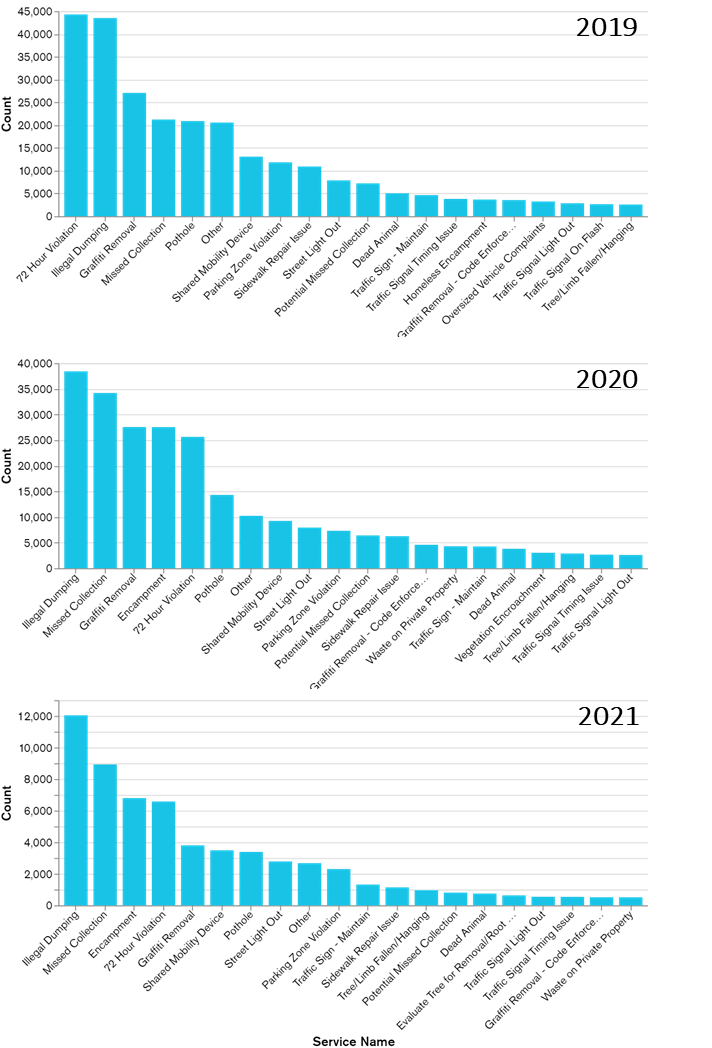
In 2019, service requests peeked in July and August. District data should be verified to see if spikes could possibly occur in more popular tourist areas.

In 2020, service requests dipped significantly in April which correlates with the start of COVID-19 restrictions, and peeked in October.

## Service Types

When looking at the annual summary statistics, the following is noted:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Type** | **2019** | **2020** | **2021** | **Trend** |
| Illegal Dumping | 44K | 38K | 12K | 🡅 |
| Missed Collection | 21K | 34K | 9K | 🡅 |
| Encampment | 4K | 27K | 7K | ⯇⯈ |

Since 2019, illegal dumping has been the most popular service request in the City of San Diego. Further analysis is required to see if locations of the occurrences are repetitive or clustered.

Missed collections is experiencing a significant increase since 2019 and is still trending upwards. Council district data should be reviewed to determine if this is a city-wide trend or if this is occurring in specific areas. Causes could be things such as an aging vehicle fleet or area population growth. Possible remedies could be additional training if the issue is area-specific, additional resources if the issue is city-wide.

## Council Districts Performance

|  |  |  |  |
| --- | --- | --- | --- |
| **Council Districts** | **2019** | **2020** | **2021 to date** |
| Most service requests | District 3 (75K) | District 3 (66K) | District 3 (25K) |
| Fewest service requests | District 5 (10K) | District 5 (10K) | District 5 (3K) |
| Average case length across districts | 46-60 days | 20-35 days | 10-16 days |
| Longest average case length | District 5 (70 days+) | District 5 (40 days+) | District 5 (21 days+) |

The data shows that the Council District with the fewest service requests has the longest average case length. Further review disclosed that District 5 has several long-duration service request types that possibly skew the average case length:

* Evaluate for resurfacing: 280 cases at 400 days to close
* Sidewalk repair issues: 750 cases at 200 days to close

It is recommended to remove the above items from the data and reevaluate the average case length of District 5. If the data still shows a significant higher average case length, possible culprits such as high volume service requests should be reviewed to determine if there are training or resource issues.

# Going forward

The dashboard could be expanded with the following data:

* Volume of service requests by category by year / quarter / area
  + Is volume of service requests evenly distributed by area?
  + Do certain areas have a noticeably smaller or larger number of service requests in any category?
  + Is the volume of service requests steady or are there dips/spikes?
* Are dips/spikes occurring during certain times of the year? Certain districts?
  + Average days from open to close, grouped by category, by council district compared to overall
  + Is average time to resolve a service request same across categories? Across districts?
  + Dips/spikes?  Certain areas?  Certain times of year?
* Group the data by major categories (less then 20) instead of every category

# code

## Flask Setup

# Flask Setup

#################################################

app = Flask(\_\_name\_\_)

CORS(app)

#################################################

# Database Setup

#################################################

mongo = pymongo.MongoClient(config.mongo\_conn, maxPoolSize=50, connect=False)

db = pymongo.database.Database(mongo, config.db\_name)

col = pymongo.collection.Collection(db, 'sandiego')

collection\_summary = pymongo.collection.Collection(db, 'summary\_counts')

# create route that renders index.html template

@app.route("/")

def home():

    return render\_template("index.html")

@app.route("/councildistricts")

def councildistricts():

  return render\_template("councildistricts.html")

@app.route("/api/data")

def data():

    results = json.loads(dumps(col.find().limit(500).sort("time", -1)))

    return jsonify(results)

@app.route("/api/daterequested/<year>!<name>!<limit>")

def daterequested(year, name, limit):

    year\_int = 2021

    limit\_int = 1000

    try:

        year\_int = int(year)

        limit\_int = int(limit)

    except ValueError:

        # Handle the exception

        "Invalid Year"

    local = pytz.timezone("America/Los\_Angeles")

    dt\_start = datetime.strptime(str(year\_int) + "-1-1 00:00:00", "%Y-%m-%d %H:%M:%S")

    dt\_start\_local = local.localize(dt\_start, is\_dst=None)

    dt\_start\_utc = dt\_start\_local.astimezone(pytz.utc)

    dt\_end = datetime.strptime(str(year\_int + 1) + "-1-1 00:00:00", "%Y-%m-%d %H:%M:%S")

    dt\_end\_local = local.localize(dt\_end, is\_dst=None)

    dt\_end\_utc = dt\_end\_local.astimezone(pytz.utc)

    print(dt\_start\_utc, dt\_end\_utc)

    if name != "All":

        filter={

            'date\_requested': {

                '$gte': dt\_start\_utc,

                '$lt': dt\_end\_utc

            },

            'service\_name':name

        }

    else:

        filter={

            'date\_requested': {

                '$gte': dt\_start\_utc,

                '$lt': dt\_end\_utc

            }

        }

    results = json.loads(dumps(col.find(filter=filter).limit(limit\_int).sort("time", -1)))

    return jsonify(results)

@app.route("/api/summary/<year>")

def summary(year):

    filter= {"year":int(year)}

    results = json.loads(dumps(collection\_summary.find(filter=filter)))

    return jsonify(results)

@app.route("/api/cdSummary/<district>")

def cdSummary(district):

    filter= {"district":int(district)}

    results = json.loads(dumps(collection\_summary.find(filter=filter)))

    return jsonify(results)

@app.route("/api/servicenames")

def servicenames():

    results = json.loads(dumps(col.distinct("service\_name")))

    return jsonify(results)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, port=5104)

## Main\_mbm.jpynb to process the data

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Timeline

Description automatically generated with low confidence

## Visualization.py for the summary statistics

Graphical user interface, text, application, email

Description automatically generated

## App.py to populate main dashboard

d3.selectAll("body").on("change", populateDashboard);

function populateServicesNames() {

  url\_servicenames = "api/servicenames";

  d3.json(url\_servicenames).then(function(response) {

    console.log(response)

    var serviceNameArr = response

    // select inputs

    var inputElementDate = d3.select("#selServiceName");

    // auto populate available filter days and add blank option to search without date filter

    serviceNameArr.forEach(servicename => {

         inputElementDate.append('option').text(servicename);

    });

  });

}

function populateDashboard() {

  console.log("loading summary data...")

  // Use D3 to select the dropdown menu

  var CB\_Year = d3.select("#selYear");

  // Assign the value of the dropdown menu option to a variable

  var year = CB\_Year.node().value;

  console.log(year);

  /\* data route \*/

  const url = "api/summary" + "/" + year;

  d3.json(url).then(function(response) {

    /////////////////////////////////////////////////

    // Multiline Plot SR over time

    /////////////////////////////////////////////////

    console.log(year);

    const data = response[0].summary;

    //console.log(data);

    month = []

    count = []

    for (const [key, value] of Object.entries(data)) {

      // console.log(`${key}: ${value}`);

      month.push(key);

      count.push(value);

    }

    countbymonth = [{

      x: month,

      y: count }];

    var lineplot = d3.selectAll("#line-plot").node();

    Plotly.newPlot(lineplot, countbymonth);

    console.log("Summary data loaded.")

    // bar chart count by service request type

    var chart\_url = response[0].chart\_url;

    chart\_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-radius: 2px;box-shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart\_url + "'></iframe>"

    d3.select("#bar-plot").html(chart\_html)

    // bar chart count by council district

    var chart\_url\_council\_dist = response[0].chart\_url\_council\_dist

    chart\_council\_dist\_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-radius: 2px;box-shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart\_url\_council\_dist + "'></iframe>"

    d3.select("#bar-plot-council-dist").html(chart\_council\_dist\_html)

    var sr\_name = "All"

    var limit = 1000

    // Use D3 to select the dropdown menu

    var CB\_SRName = d3.select("#selServiceName");

    // Assign the value of the dropdown menu option to a variable

    sr\_name = CB\_SRName.node().value;

    // Use D3 to select the dropdown menu

    var CB\_Limit = d3.select("#selMapLimit");

    // Assign the value of the dropdown menu option to a variable

    limit = CB\_Limit.node().value;

    console.log(year, sr\_name, limit);

    map\_html = "<iframe src='http://127.0.0.1:5500/pages/index.html?year=" + year + "&name=" + sr\_name + "&limit=" + limit + "' height='600px' width='100%' title='Service Request Cluster Map'></iframe>";

    d3.select("#map").html(map\_html)

    d3.select("#total\_requested").html("<span>" + response[0].total\_requested + "</span>")

    d3.select("#total\_closed").html("<span>" + response[0].total\_closed + "</span>")

    d3.select("#percent\_closed").html("<span>" + (parseFloat(response[0].percent\_closed)\*100).toFixed(2).toString() + "%</span>")

    d3.select("#average\_case\_age").html("<span>" + parseFloat(response[0].average\_case\_age\_days).toFixed(2).toString() + "</span>")

  });

}

function populateDistricts() {

  console.log("loading summary data...")

  // Use D3 to select the dropdown menu

  var CB\_District = d3.select("#selDistrict");

  // Assign the value of the dropdown menu option to a variable

  var district = CB\_District.node().value;

  console.log(district);

  /\* data route \*/

  const cdurl = "api/summary" + "/" + district;

  d3.json(cdurl).then(function(response) {

    console.log(district);

    var line\_url = response[0].chart\_url\_cd\_tickets\_over\_time;

    line\_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-radius: 2px;box-shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + line\_url + "'></iframe>"

    d3.select("#cd-line-plot").html(line\_html)

    // bar chart count by service request type

    var chart\_url = response[0].chart\_url;

    chart\_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-radius: 2px;box-shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart\_url + "'></iframe>"

    d3.select("#cd-bar-plot").html(chart\_html)

    // bar chart count by council district

    var chart\_url\_council\_dist = response[0].chart\_url\_council\_dist

    chart\_council\_dist\_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-radius: 2px;box-shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart\_url\_council\_dist + "'></iframe>"

    d3.select("#bar-plot-council-dist").html(chart\_council\_dist\_html)

    d3.select("#total\_requested").html("<span>" + response[0].total\_requested + "</span>")

    d3.select("#total\_closed").html("<span>" + response[0].total\_closed + "</span>")

    d3.select("#percent\_closed").html("<span>" + (parseFloat(response[0].percent\_closed)\*100).toFixed(2).toString() + "%</span>")

    d3.select("#average\_case\_age").html("<span>" + parseFloat(response[0].average\_case\_age\_days).toFixed(2).toString() + "</span>")

  });

}

function init() {

  populateServicesNames();

  populateDashboard();

  populateDistricts();

};

init();

## Cdapp.js to populate council districts dashboard

d3.selectAll("body").on("change", populateDistricts);

function populateDistricts() {

  console.log("loading summary data...")

  // Use D3 to select the dropdown menu

  var CB\_District = d3.select("#selDistrict");

  // Assign the value of the dropdown menu option to a variable

  var district = CB\_District.node().value;

  console.log(district);

  /\* data route \*/

  const cdurl = "api/cdSummary" + "/" + district;

  d3.json(cdurl).then(function(response) {

    console.log(district);

    var line\_url = response[0].chart\_url\_cd\_tickets\_over\_time;

    line\_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-radius: 2px;box-shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + line\_url + "'></iframe>"

    d3.select("#cd-line-plot").html(line\_html)

    // bar chart count by service request type

    var chart\_url = response[0].chart\_url;

    chart\_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-radius: 2px;box-shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart\_url + "'></iframe>"

    d3.select("#cd-bar-plot").html(chart\_html)

    // bar chart count by council district

    var chart\_url\_council\_dist = response[0].chart\_url\_cd\_average\_case\_length;

    chart\_council\_dist\_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-radius: 2px;box-shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart\_url\_council\_dist + "'></iframe>"

    d3.select("#bar-plot-council-dist").html(chart\_council\_dist\_html)

    d3.select("#total\_requested").html("<span>" + response[0].total\_requested + "</span>")

    d3.select("#total\_closed").html("<span>" + response[0].total\_closed + "</span>")

    d3.select("#percent\_closed").html("<span>" + (parseFloat(response[0].percent\_closed)\*100).toFixed(2).toString() + "%</span>")

    d3.select("#average\_case\_age").html("<span>" + parseFloat(response[0].average\_case\_age\_days).toFixed(2).toString() + "</span>")

  });

}

function init() {

  populateDistricts();

};

init();

## Logic.JS to create map

// Creating map object

var myMap = L.map("map", {

  center: [32.7157, -117.1611],

  zoom: 11

});

// Adding tile layer to the map

L.tileLayer("https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access\_token={accessToken}", {

  attribution: "© <a href='https://www.mapbox.com/about/maps/'>Mapbox</a> © <a href='http://www.openstreetmap.org/copyright'>OpenStreetMap</a> <strong><a href='https://www.mapbox.com/map-feedback/' target='\_blank'>Improve this map</a></strong>",

  tileSize: 512,

  maxZoom: 18,

  zoomOffset: -1,

  id: "mapbox/streets-v11",

  accessToken: API\_KEY

}).addTo(myMap);

const urlParams = new URLSearchParams(window.location.search);

// Store API query variables

var baseURL = "http://127.0.0.1:5104/api/daterequested/";

var year = urlParams.get('year');

var sr\_name = "!" + urlParams.get('name');

var limit = "!" + urlParams.get('limit');

console.log(year);

// Assemble API query URL

var url = baseURL + year + sr\_name + limit

console.log(url);

// Grab the data with d3

d3.json(url, function(response) {

  // Create a new marker cluster group

  var markers = L.markerClusterGroup();

  // Loop through data

  for (var i = 0; i < response.length; i++) {

      lat = response[i]["lat"]

      lng = response[i]["lng"]

      var details = "service\_request\_id: " + response[i]["service\_request\_id"];

      if (response[i]["date\_requested"]) {

        details += "<br>date\_requested: " + response[i]["date\_requested"]

        console.log(response[i]["date\_requested"]);

      } if (response[i]["case\_age\_days"]) {

        details += "<br>case\_age\_days: " + response[i]["case\_age\_days"]

      } if (response[i]["service\_name"]) {

        details += "<br>service\_name: " + response[i]["service\_name"]

      } if (response[i]["case\_record\_type"]) {

        details += "<br>case\_record\_type: " + response[i]["case\_record\_type"]

      } if (response[i]["date\_closed"]) {

        details += "<br>date\_closed: " + response[i]["date\_closed"]

        console.log(response[i]["date\_closed"]);

      } if (response[i]["status"]) {

        details += "<br>status: " + response[i]["status"]

      } if (response[i]["street\_address"]) {

        details += "<br>street\_address: " + response[i]["street\_address"]

      } if (response[i]["council\_district"]) {

        details += "<br>council\_district: " + response[i]["council\_district"]

      } if (response[i]["comm\_plan\_code"]) {

        details += "<br>comm\_plan\_code: " + response[i]["comm\_plan\_code"]

      } if (response[i]["comm\_plan\_name"]) {

        details += "<br>comm\_plan\_name: " + response[i]["comm\_plan\_name"]

      } if (response[i]["case\_origin"]) {

        details += "<br>case\_origin: " + response[i]["case\_origin"]

      } if (response[i]["public\_description"]) {

        details += "<br>public\_description: " + response[i]["public\_description"]

      } if (response[i]["media\_url"]) {

        details += "<img src='" + response[i]["media\_url"] + "' style='height:200px;float:right'/>";

      }

     // Check for location property

     if ((lat) && (lng)) {

      // Add a new marker to the cluster group and bind a pop-up

      markers.addLayer(L.marker([lat, lng])

        .bindPopup(details));

    }

   }

  // // Add our marker cluster layer to the map

  myMap.addLayer(markers);

});

## Charts

Charts were created in MondoDB charts. The links were stored in a MongoDb collection and pulled into the dashboard from there.