By **Vitor Freitas**

I'm a passionate software developer and researcher. I write about Python, Django and Web Development on a weekly basis. Read more.

TUTORIAL

How to Use Chart.js with Django

Jan 19, 2020 · 10 minutes read · 22 comments · 12,118 views

(Picture: https://www.pexels.com/photo/black-samsung-tablet-computer-106344/)

Chart.js is a cool open source JavaScript library that helps you render HTML5 charts. It is responsive and counts with 8 different chart types.

In this tutorial we are going to explore a little bit of how to make Django talk with Chart.js and render some simple charts based on data extracted from our models.

Installation

For this tutorial all you are going to do is add the Chart.js lib to your HTML page:

```
<script src="https://cdn.jsdelivr.net/npm/chart.js@2.9.3/dist/Chart.min.js"></script>
```

You can download it from Chart.js official website and use it locally, or you can use it from a CDN using the URL above.

Example Scenario

I'm going to use the same example I used for the tutorial How to Create Group By Queries With Django ORM which is a good complement to this tutorial because actually the tricky part of working with charts is to transform the data so it can fit in a bar chart / line chart / etc.

We are going to use the two models below, `Country` and `City`:

```
class Country(models.Model):
    name = models.CharField(max_length=30)

class City(models.Model):
    name = models.CharField(max_length=30)
    country = models.ForeignKey(Country, on_delete=models.CASCADE)
    population = models.PositiveIntegerField()
```

And the raw data stored in the database:

id	name	country_id	population
1	Tokyo	28	36,923,000
2	Shanghai	13	34,000,000
3	Jakarta	19	30,000,000
4	Seoul	21	25,514,000
5	Guangzhou	13	25,000,000

id	name
1	Brazil
2	Turkey
3	Italy
4	Bangladesh
5	Canada

Example 1: Pie Chart

For the first example we are only going to retrieve the top 5 most populous cities and render it as a pie chart. In this strategy we are going to return the chart data as part of the view context and inject the results in the JavaScript code using the Django Template language.

views.py

```
from django.shortcuts import render
from mysite.core.models import City

def pie_chart(request):
    labels = []
    data = []

    queryset = City.objects.order_by('-population')[:5]
    for city in queryset:
        labels.append(city.name)
        data.append(city.population)

    return render(request, 'pie_chart.html', {
        'labels': labels,
        'data': data,
    })
```

Basically in the view above we are iterating through the `City` queryset and building a list of `labels` and a list of `data`. Here in this case the `data` is the population count saved in the `City` model.

For the `urls.py` just a simple routing:

urls.py

```
from django.urls import path
from mysite.core import views

urlpatterns = [
    path('pie-chart/', views.pie_chart, name='pie-chart'),
]
```

Now the template. I got a basic snippet from the Chart.js Pie Chart Documentation.

pie_chart.html

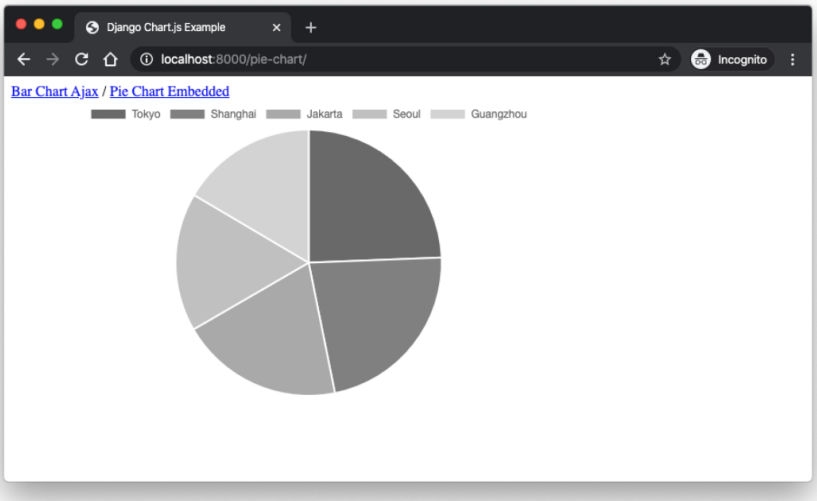
```
{% extends 'base.html' %}

{% block content %}
  <div id="container" style="width: 75%;">
    <canvas id="pie-chart"></canvas>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/chart.js@2.9.3/dist/Chart.min.js"></script>
  <script>

    var config = {
      type: 'pie',
      data: {
        datasets: [{
          data: {{ data|safe }},
          backgroundColor: [
            '#696969', '#808080', '#A9A9A9', '#C0C0C0', '#D3D3D3'
          ],
          label: 'Population'
        }],
        labels: {{ labels|safe }}
      },
      options: {
```

```
        responsive: true
      }
    };

    window.onload = function() {
      var ctx = document.getElementById('pie-chart').getContext('2d');
      window.myPie = new Chart(ctx, config);
    };

  </script>

{% endblock %}
```

In the example above the `base.html` template is not important but you can see it in the code example I shared in the end of this post.

This strategy is not ideal but works fine. The bad thing is that we are using the Django Template Language to interfere with the JavaScript logic. When we put `{{ data|safe}}` we are injecting a variable that came from the server directly in the JavaScript code.

The code above looks like this:



Example 2: Bar Chart with Ajax

As the title says, we are now going to render a bar chart using an async call.

views.py

```
from django.shortcuts import render
from django.db.models import Sum
from django.http import JsonResponse
from mysite.core.models import City

def home(request):
    return render(request, 'home.html')

def population_chart(request):
    labels = []
    data = []

    queryset = City.objects.values('country__name').annotate(country_population=Sum('population')).order
    for entry in queryset:
        labels.append(entry['country__name'])
        data.append(entry['country_population'])

    return JsonResponse(data={
        'labels': labels,
        'data': data,
    })
```

So here we are using two views. The `home` view would be the main page where the chart would be loaded at. The other view `population_chart` would be the one with the sole responsibility to aggregate the data the return a JSON response with the labels and data.

If you are wondering about what this queryset is doing, it is grouping the cities by the country and aggregating the total population of each country. The result is going to be a list of country + total population. To learn more about this kind of query have a look on this post: How to Create Group By Queries With Django ORM

urls.py

```
from django.urls import path
from mysite.core import views

urlpatterns = [
    path('', views.home, name='home'),
    path('population-chart/', views.population_chart, name='population-chart'),
]
```

home.html

```
{% extends 'base.html' %}

{% block content %}

  <div id="container" style="width: 75%;">
    <canvas id="population-chart" data-url="{% url 'population-chart' %}"></canvas>
  </div>

  <script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@2.9.3/dist/Chart.min.js"></script>
  <script>

    $(function () {

      var $populationChart = $("#population-chart");
      $.ajax({
        url: $populationChart.data("url"),
        success: function (data) {

          var ctx = $populationChart[0].getContext("2d");

          new Chart(ctx, {
            type: 'bar',
            data: {
              labels: data.labels,
              datasets: [{
                label: 'Population',
                backgroundColor: 'blue',
                data: data.data
              }]
            },
            options: {
              responsive: true,
              legend: {
                position: 'top',
              },
              title: {
                display: true,
                text: 'Population Bar Chart'
              }
            }
          });

        }
      });

    });

  </script>

{% endblock %}
```

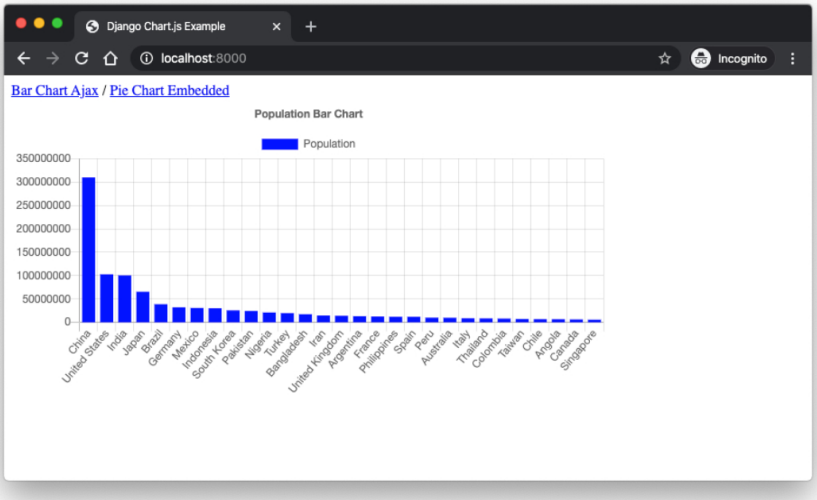
Now we have a better separation of concerns. Looking at the chart container:

```
<canvas id="population-chart" data-url="{% url 'population-chart' %}"></canvas>
```

We added a reference to the URL that holds the chart rendering logic. Later on we are using it to execute the Ajax call.

```
var $populationChart = $("#population-chart");
$.ajax({
  url: $populationChart.data("url"),
  success: function (data) {
    // ...
  }
});
```

Inside the `success` callback we then finally execute the Chart.js related code using the `JsonResponse` data.



Conclusions

I hope this tutorial helped you to get started with working with charts using Chart.js. I published another tutorial on the same subject a while ago but using the Highcharts library. The approach is pretty much the same: How to Integrate Highcharts.js with Django.

If you want to grab the code I used in this tutorial you can find it here: github.com/sibtc/django-chartjs-example.

Related Posts

How to Integrate Highcharts.js with Django

django charts chartjs

Share this post

Walison Filipe ➚ Caio • a year ago

yeah, make sense

^ | v • Reply • Share ›

Z Ren ➚ Walison Filipe • a year ago

No it doesn't. django.urls.path was introduced in Django 2, too. So the examples wouldn't work in older versions anyways.

1 ^ | v • Reply • Share ›

ram0973 • a year ago • edited

Please write an article, how simple CMS work? For example:
I'm doing a site with Django. I can do 2 modules: news and pages. Every module implement abstract SEO Mixin and MPTT Category Tree. And I stuck here - how to do SEO for home page, for example? Make a page 'home' in pages module?

Other way is to do any SiteTree module, which includes SEO and tree categories, and somehow attach news and pages (without SEO and MPTT - much simpler modules) to SiteTree. As a benefit, I easily can get a Menus and Breadcrumbs from this module. And a sitemap.xml maybe. I think this is how Wagtail CMS works.

Can you give an idea how to do this without Django CMS, FeinCMS, Mezzanine or something?
I tried them, but I don't understand what is going on under their hood.
In other words, how perfect CMS must do this (site tree with SEO, Categories)?

And how to extend it, if I want, for example, to make a page with forms?

1 ^ | v 1 • Reply • Share ›