Embed pygal charts in Django Application

Last Updated : 26 Sep, 2023

Suppose we are developing a web application using the Django framework and we have some data and want to visualize it on the webpage We can embed it in Django templates and render it to the web browser using Python.

Embed Pygal Charts in Django Application

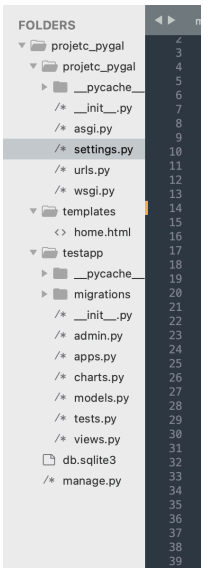
Embedding Pygal charts in a Django template involves integrating dynamic charts created with Pygal, a Python charting library, into your Django web application's front end. This process lets you visualize data in a visually appealing way directly on your website.

Required Installation

Command to install Django and Pygal.

```
pip3 install django
pip install pygal
```

File Structure



Steps to Embed Charts in Django

Step 1: First make a project by using the command :

```
Django-admin startproject  project_pygal
```

Step 2: Create a application named ‘testapp’ by using the command :

```
python3 manage.py startapp testapp
```

Creating Necessary Files

charts.py: This Python code defines three classes, EmployeePieChart, EmployeeGaugeChart, and EmployeeBarChart, which generate different types of charts (Pie, Line, and Bar charts) using the Pygal library. These classes fetch data from the Django Employee model, specifically department-wise employee strength, and create charts based on this data. Each class has an __init__ method to set up the chart type and title, a get_data method to retrieve data from the model, and a generate method to generate and render the chart. These classes provide a convenient way to visualize employee data in different chart formats within a Django application.

Python3

```
import pygal
from .models import Employee
class EmployeePieChart():
    def __init__(self,**kwargs):
        self.chart=pygal.Pie(**kwargs)
        self.chart.title='Employees in different department'
    def get_data(self):
        data={}
        for emp in Employee.objects.all():
            data[emp.department]=emp.strength
        return data
    def generate(self):
        chart_data=self.get_data()
        for key,value in chart_data.items():
            self.chart.add(key,value)
        return self.chart.render(is_unicode=True)
class EmployeeGaugeChart():
    def __init__(self,**kwargs):
        self.chart=pygal.Gauge(**kwargs)
        self.chart.title='Employees in different department'
```

```
            for emp in Employee.objects.all():
                data[emp.department]=emp.strength
            return data
        def generate(self):
            chart_data=self.get_data()
            for key,value in chart_data.items():
                self.chart.add(key,value)
            return self.chart.render(is_unicode=True)
class EmployeeBarChart():
    def __init__(self,**kwargs):
        self.chart=pygal.Bar(**kwargs)
        self.chart.title='Employees in different department'
    def get_data(self):
        data={}
        for emp in Employee.objects.all():
            data[emp.department]=emp.strength
        return data
    def generate(self):
        chart_data=self.get_data()
        for key,value in chart_data.items():
            self.chart.add(key,value)
        return self.chart.render(is_unicode=True)
```

views.py: This Django code defines views to manage and visualize employee data. The clear view deletes all employee records and redirects to the home page. The home view handles adding new employee data. The three IndexView views render the home page with different chart types (Pie, Gauge, and Bar) generated using Pygal. These views fetch data from the Employee model and use specific chart classes to create and display dynamic charts. This setup allows users to interact with employee data and view it in various chart formats within a Django application.

Python3

```
from django.views.generic import TemplateView
from pygal.style import DarkStyle
from django.shortcuts import render,redirect
from django.http import HttpResponse
from testapp.models import Employee
from .charts import EmployeePieChart,EmployeeGaugeChart,EmployeeBarChart

def clear(request):
```

```
    Employee.objects.all().delete()
    return redirect('/home')
def home(request):
    if request.method=='POST':
        department=request.POST['department']
        strength=request.POST['strength']
        print(department)
        print(strength)
        Employee.objects.create(department=department, strength=strength)


    return render(request,'home.html')

class IndexView(TemplateView):
    template_name = 'index.html'

    def get_context_data(self, **kwargs):
        context = super(IndexView, self).get_context_data(**kwargs)

        # Instantiate our chart. We'll keep the size/style/etc.
        # config here in the view instead of `charts.py`.
        cht_employee = EmployeePieChart(
            height=600,
            width=800,
            explicit_size=True,
            style=DarkStyle
        )

        # Call the `.generate()` method on our chart object
        # and pass it to template context.
        context['cht_employee'] = cht_employee.generate()
        return context
class IndexView1(TemplateView):
    template_name = 'index.html'

    def get_context_data(self, **kwargs):
        context = super(IndexView1, self).get_context_data(**kwargs)

        # Instantiate our chart. We'll keep the size/style/etc.
        # config here in the view instead of `charts.py`.
        cht_employee = EmployeeLineChart(
            height=600,
            width=800,
            explicit_size=True,
            style=DarkStyle
        )

        # Call the `.generate()` method on our chart object
        # and pass it to template context.
```

```
        context['cht_employee'] = cht_employee.generate()
        return context
class IndexView2(TemplateView):
    template_name = 'index.html'

    def get_context_data(self, **kwargs):
        context = super(IndexView2, self).get_context_data(**kwargs)

        # Instantiate our chart. We'll keep the size/style/etc.
        # config here in the view instead of `charts.py`.
        cht_employee = EmployeeBarChart(
            height=600,
            width=800,
            explicit_size=True,
            style=DarkStyle
        )

        # Call the `.generate()` method on our chart object
        # and pass it to template context.
        context['cht_employee'] = cht_employee.generate()
        return context
```

models.py: This Django model, named Employee, has two fields: department to store the employee's department as text and strength to store the number of employees in that department as an integer. The __str__ method determines how an Employee instance is represented as a string.

Python3

```
from django.db import models

class Employee(models.Model):
    department=models.CharField(max_length=255)
    strength=models.IntegerField()


    def __str__(self):
        return self.department
```

Setting up GUI

home.html: This HTML code creates a form for users to input department and strength data. It includes labels and input fields, a “Submit” button, and styled hyperlinks functioning as buttons to navigate to different views within a Django application, including charts and data clearing options. The buttons have hover effects for visual appeal.

HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Department and Strength Form</title>
    <style>
        /* Define styles for the box-style button */
        .btn {
            display: inline-block;
            padding: 10px 20px;
            background-color: #007bff;
            color: #fff;
            text-decoration: none;
            border: 2px solid #007bff;
            border-radius: 5px;
            font-size: 16px;
            cursor: pointer;
            margin-top: 20px; /* Add some space below the form */
        }
        .btn:hover {
            background-color: #0056b3;
            border-color: #0056b3;
        }
    </style>
</head>
<body>
    <h1>Enter Details</h1>
    <form method="post">
        {% csrf_token %}
        <label for="department">Department:</label>
        <input type="text" name="department"><br>
        <label for="strength">Strength:</label>
        <input type="number" name="strength"><br>

        <input type="submit" value="Submit">
    </form>
    <a href="/index" class="btn">Click to Pie chart</a>
    <a href="/index1" class="btn">Click to  Gauge chart</a>
    <a href="/index2" class="btn">Click to Bar chart</a>
```

```
    <a href="/clear" class="btn">Click to Clear data</a>
</body>
</html>
```

index.html: This HTML template is designed to display Pygal charts generated by Django views. It includes the chart using a Django template variable and loads a Pygal tooltips script for improved chart interactivity.

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>django-pygal</title>
</head>
<body>
  {{ cht_employee|safe }}
  <script type="text/javascript" src="http://kozea.github.com/pygal.js/latest/pyga
</body>
</html>
```

urls.py: This Django URL configuration defines the URL patterns for routing requests to specific views. It includes routes for the admin interface, views to display charts, and a view to clear data.

Python3

```
"""projetc_pygal URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
```

```
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from testapp.views import IndexView,home,IndexView1,IndexView2,clear

urlpatterns = [
    path('admin/', admin.site.urls),
    path('home/', home),
    path('index/', IndexView.as_view()),
    path('index1/', IndexView1.as_view()),
    path('index2/', IndexView2.as_view()),
    path('clear/', clear),

]
```

Deployement of the Project

Run these commands to apply the migrations:

```
python3 manage.py makemigrations
python3 manage.py migrate
```

Run the server with the help of following command:

```
python3 manage.py runserver
```

Output Video:

Media error: Format(s) not supported or source(s) not found

mejs.download-file: https://media.geeksforgeeks.org/wp-content/uploads/20230825114226//Department-and-Strength-Form-(1).mp4?_=1

k -

rs

ld

r

you're just starting or looking to refine your skills, this course will empower you to build sophisticated web applications efficiently. Ready to enhance your web development journey? Enroll now and unlock your potential with Django!

M maya...

Previous Article

Filter Objects With Count Annotation in Django

Next Article

Similar Reads

Solid Gauge Chart in Pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

1 min read

Pyramid chart in pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

2 min read

Gauge Chart in pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

2 min read

Radar chart in pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

2 min read

Histogram in Pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

2 min read

Half pie chart in Pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

2 min read

Treemap in Pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

2 min read

Funnel Chart in Pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

2 min read

Bar Chart in Pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

2 min read

Stacked Bar chart in pygal

Pygal is a Python module that is mainly used to build SVG (Scalar Vector Graphics) graphs and charts. SVG is a vector-based graphics in the XML format...

2 min read

Article Tags : Django Python Django Python pygal-chart

Corporate & Communications Address:- A-143, 9th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305) | Registered Address:- K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305