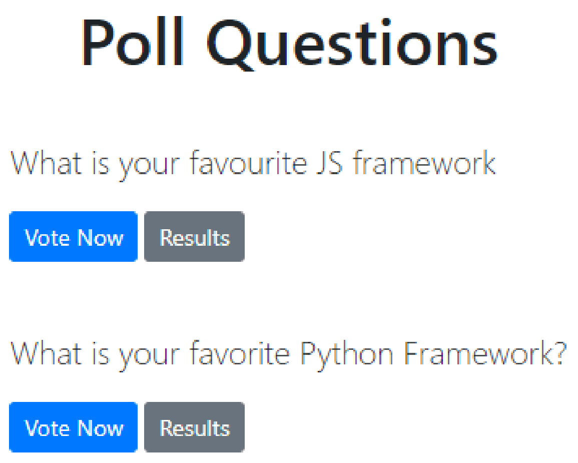
Voting System Project Using Django Framework

Last Updated : 09 Sep, 2024

Project Title: Pollster (Voting System) web application using Django framework
Type of Application (Category): Web application.

Introduction: We will create a pollster (voting system) web application using Django. This application will conduct a series of questions along with many choices. A user will be allowed to vote for that question by selecting a choice. Based on the answer the total votes will be calculated and it will be displayed to the user. Users can also check the result of the total votes for specific questions on the website directly. We will also build the admin part of this project. Admin user will be allowed to add questions and manage questions in the application.


Poll Questions
What is your favourite JS framework
Vote Now
Results
What is your favorite Python Framework?
Vote Now
Results

Pre-requisite: Knowledge of Python and basics of Django Framework. Python should be installed in the system. Visual studio code or any code editor to work on the application.

Technologies used in the project: Django framework and SQLite database which comes by default with Django.

Implementation of the Project

Creating Project

Step-1: Create an empty folder **pollster_project** in your directory.

Step-2: open your command prompt using window+r key

Step-2: Now switch to your folder using

cd pollster_project

cd pollster_project

step-4: and create a virtual environment in this folder using the following command.

```
pip install pipenv
```

```
pipenv shell
```

Step-3: A **Pipfile** will be created in your folder from the above step. Now install Django in your folder using the following command.

```
pipenv install django
```

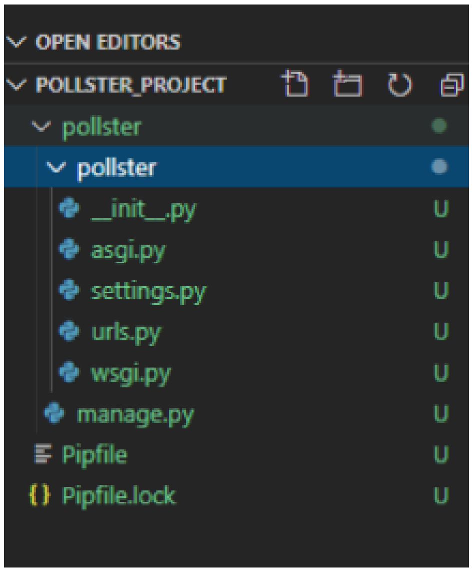
Step-4: Now we need to establish the Django project. Run the following command in your folder and initiate a Django project.

```
django-admin startproject pollster
```

A New Folder with name **pollster** will be created. Switch to the pollster folder using the following command.

```
cd pollster
```

The folder structure will look something like this.



```
OPEN EDITORS
POLLSTER_PROJECT
  pollster
    pollster
      __init__.py      U
      asgi.py          U
      settings.py      U
      urls.py          U
      wsgi.py          U
    manage.py          U
  Pipfile              U
  Pipfile.lock         U
```

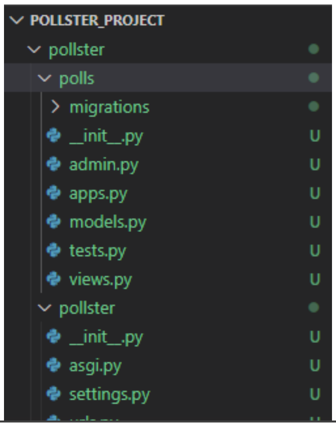
Here you can start the server using the following command and check if the application running or not using your **http://127.0.0.1:8000/** in your browser.

```
python manage.py runserver
```

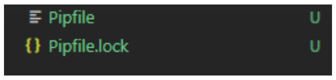
Step-5: Create an app ‘**polls**‘ using the following command

```
python manage.py startapp polls
```

Below is the folder structure after creating ”polls’ app in the project.



```
POLLSTER_PROJECT
  pollster
    polls
      migrations
      __init__.py      U
      admin.py         U
      apps.py          U
      models.py        U
      tests.py         U
      views.py         U
    pollster
      __init__.py      U
      asgi.py          U
      settings.py      U
```



```
  Pipfile              U
  Pipfile.lock         U
```

Create Models

Step-1: In your **models.py** file write the code given below to create two tables in your database. One is ‘**Question**‘ and the other one is ‘**Choice**‘. ‘Question’ will have two fields of ‘question_text’ and a ‘pub_date’. Choice has three fields: ‘question’, ‘choice_text’, and ‘votes’. Each Choice is associated with a Question.

Python

```python
from django.db import models

# Create your models here.


class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

    def __str__(self):
        return self.question_text


class Choice(models.Model):
    question = models.ForeignKey(Question, 
on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

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

Step-2: Go to the **settings.py** file and in the list, INSTALLED_APPS write down the code below to include the app in our project. This will refer to the polls -> apps.py -> PollsConfig class.

Python

```python
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Step-3: We have made changes in our database and created some tables but in order to reflect these changes we need to create migration here and then Django application will stores changes to our models. Run the following command given below to create migrations.

```
python manage.py makemigrations polls
```

Inside polls->migrations a file **0001_initial.py** will be created where you can find the database tables which we have created in our models.py file. Now to insert all the tables in our database run the command given below...

```
python manage.py migrate
```

Create an Admin User

Step-1: Run the command given below to create a user who can login to the admin site.

```
python manage.py createsuperuser
```

It will prompt username which we need to enter.

```
Username: geeks123
```

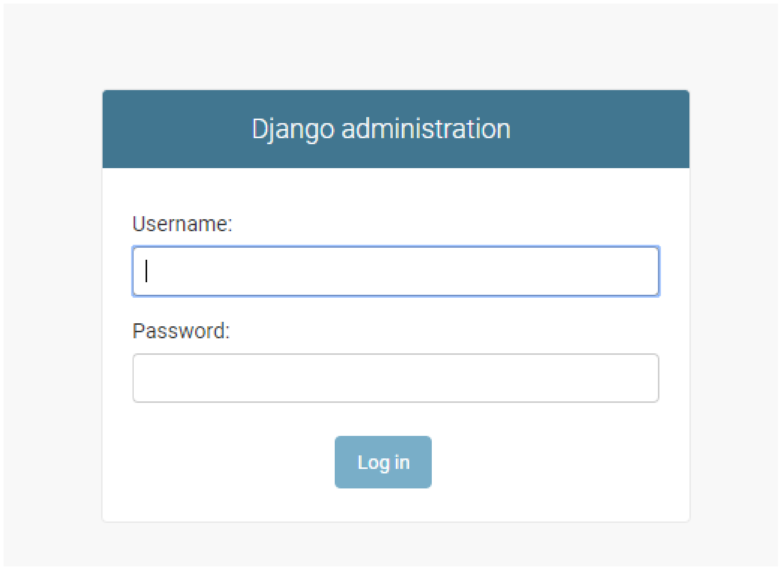
Now it will prompt an email address which again we need to enter here.

```
Email address: xyz@example.com
```

The final step is to enter the password. We need to enter the password twice, the second time as a confirmation of the first.

```
Password: ******
Password (again): ******
Superuser created successfully.
```

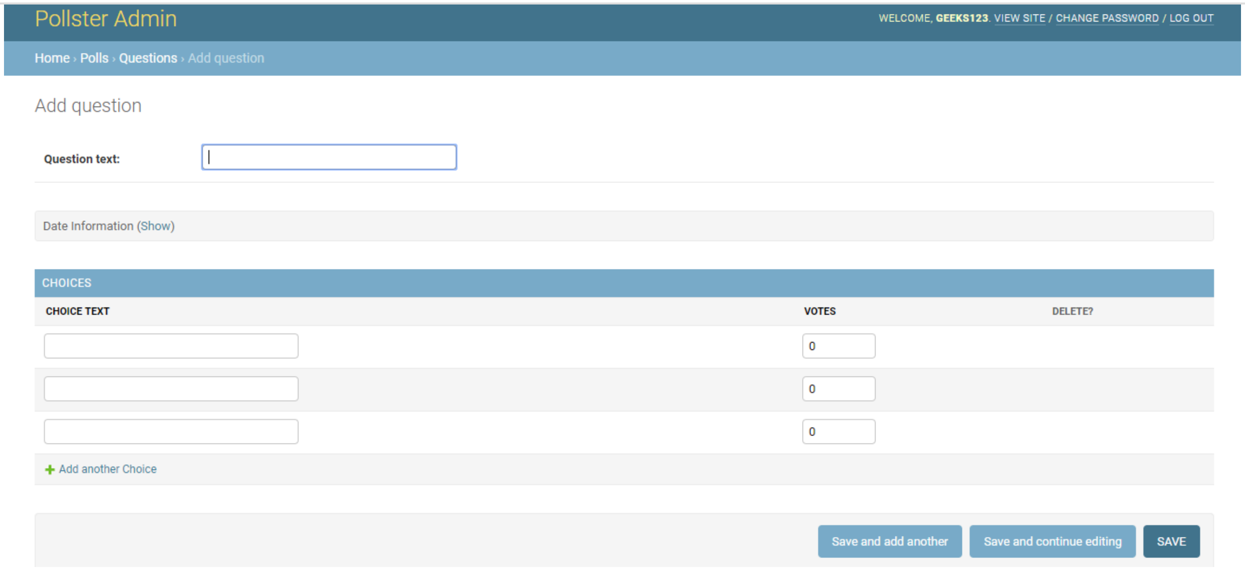
Now we can run the server using the same command **python manage.py runserver** and we can check our admin panel browsing the URL **http://127.0.0.1:8000/admin** .



Step-2: In the **admin.py** file we will write the code given below to map each question with choices to select. Also, we will write the code to change the site header, site title, and index_title. Once this is done we can add questions and choices for the question from the admin panel.

Python

```python
from django.contrib import admin
# Register your models here.
from .models import Question, Choice

# admin.site.register(Question)
# admin.site.register(Choice)

admin.site.site_header = "Pollster Admin"
admin.site.site_title = "Pollster Admin Area"
admin.site.index_title = "Welcome to the Pollster Admin
Area"


class ChoiceInLine(admin.TabularInline):
    model = Choice
    extra = 3


class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [(None, {'fields': ['question_text']}),
('Date Information', {
        'fields': ['pub_date'], 'classes': ['collapse']}), ]
        inlines = [ChoiceInLine]


admin.site.register(Question, QuestionAdmin)
```



Note: We can test the application here by adding some questions and choices for those questions.

Create Views

Now we will create the view of our application that will fetch the data from our database and will render the data in the '**template**' (we will create 'template' folder and the files inside this folder in the next section) of our application to display it to the user.

Step-1 Open **views.py** file and write down the code given below.

Python

```python
from django.template import loader
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.urls import reverse

from .models import Question, Choice

# Get questions and display them


```

```
def index(request):
    latest_question_list = Question.objects.order_by('-
pub_date')[:5]
    context = {'latest_question_list': latest_question_list}
    return render(request, 'polls / index.html', context)

# Show specific question and choices


def detail(request, question_id):
    try:
        question = Question.objects.get(pk=question_id)
    except Question.DoesNotExist:
        raise Http404("Question does not exist")
    return render(request, 'polls / detail.html',
{'question': question})

# Get question and display results


def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls / results.html',
{'question': question})

# Vote for a question choice


def vote(request, question_id):
    # print(request.POST['choice'])
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice =
question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the question voting form.
        return render(request, 'polls / detail.html', {
            'question': question,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # Always return an HttpResponseRedirect after successfully dealing
        # with POST data. This prevents data from being posted twice if a
        # user hits the Back button.
        return HttpResponseRedirect(reverse('polls:results', args=(question.id, )))
```

Step-2: Create a file **urls.py** inside the pollster->polls folder to define the routing for all the methods we have implemented in views.py file (don't get confused with the file inside the pollster->pollster->urls.py file). Below is the code of urls.py file...

Python

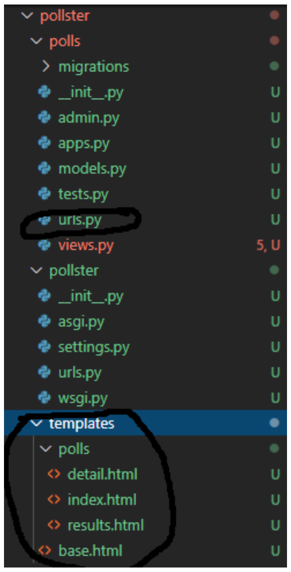
```python
from django.urls import path
from . import views

app_name = 'polls'
urlpatterns = [
    path('', views.index, name='index'),
    path('<int:question_id>/', views.detail, name='detail'),
    path('<int:question_id>/results/', views.results, name='results'),
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

Create Templates

Step-1: Follow the steps given below to create the front layout of the page.

- Create a folder '**templates**' in top-level pollster folder (alongside of polls and pollster) i.e. pollster-> templates.
- Create '**base.html**' file inside the template folder. We will define the head, body and navigation bar of our application in this file.
- In the 'templates' folder create another folder '**polls**'. In 'polls' folder create three files '**index.html**', '**results.html**' and '**detail.html**'.

The folder structure will look like the image given below (we have highlighted the files which we have created in 'create views i.e urls.py' and 'create template' section)...



Step-2: By default Django will search the 'template' inside the 'polls' app but we have created a global 'template' folder which is outside the polls app. So in order to make it work, we need to define the 'template' folder path inside the settings.py file. Open **settings.py** file and add the code given below in the list 'TEMPLATES'. In order to make the given code work add "**import os**" in settings.py.

Python

```
TEMPLATES = [
    {
        # make changes in DIRS[].
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',

'django.template.context_processors.request',

'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
        },
]
```

Step-3: Open **index.html** file and write the code given below. This file will display the **list of questions** which are stored in our database. Also, two buttons will be displayed to the user. One for the **voting** (we will create a detail.html file for voting) and the other one is to check the **results** (we will create results.html file for results).

Python

```
{% extends 'base.html' % }
{% block content % }
<h1 class = "text-center mb-3" > Poll Questions < /h1 >
{% if latest_question_list % }
{% for question in latest_question_list % }
<div class = "card-mb-3" >
   <div class = "card-body" >
        <p class = "lead" > {{question.question_text}} < /p
>
        <a href = "{% url 'polls:detail' question.id %}"
class = "btn btn-primary btn-sm" > Vote Now < /a >
        <a href = "{% url 'polls:results' question.id %}"
class = "btn btn-secondary btn-sm" > Results < /a >
    </div >
</div >
{ % endfor % }
{ % else % }
<p > No polls available < /p>
{ % endif % }
{ % endblock % }
```

Step-4: Open **detail.html** file and write the code given below. This file will be responsible for voting on specific questions. Whatever question a user will select for voting from the list of the question (index.html file), that specific question and the choices for the question will be displayed on this page. A user will be allowed to select one choice and give voting by clicking on the vote button.

Python

```
{ % extends 'base.html' % }
{ % block content % }
<a class = "btn btn-secondary btn-sm mb-3" href = "{% url
'polls:index' %}" > Back To Polls < /a >
<h1 class = "text-center mb-3" > {{question.question_text}}
< /h1 >

{ % if error_message % }
<p class = "alert alert-danger" >
<strong > {{error_message}} < /strong >
</p >
{ % endif % }

<form action = "{% url 'polls:vote' question.id %}" method =
"post" >
    { % csrf_token % }
    { % for choice in question.choice_set.all % }
    <div class = "form-check" >
    <input type = "radio" name = "choice" class = "form-
check-input" id = "choice{{ forloop.counter }}"
    value = "{{ choice.id }}" / >
    <label for = "choice{{ forloop.counter }}" >
{{choice.choice_text}} < /label >
    </div >
    { % endfor % }
    <input type = "submit" value = "Vote" class = "btn btn-
success btn-lg btn-block mt-4" / >
</form >
{ % endblock % }
```

Step-5: Open **results.html** file and write the code given below. This file will display the result of total votes on a specific question whatever question the user will select (from the index.html file) to check the result.

Python

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

```
<h1 class = "mb-5 text-center" > {{question.question_text}}
< /h1 >

<ul class = "list-group mb-5" >
    { % for choice in question.choice_set.all % }
    <li class = "list-group-item" >
    {{choice.choice_text}} < span class = "badge badge-
success float-right" > {{choice.votes}}
    vote{{choice.votes | pluralize}} < /span >
    </li >
    { % endfor % }
</ul >

<a class = "btn btn-secondary" href = "{% url 'polls:index'
%}" > Back To Polls < /a >
<a class = "btn btn-dark" href = "{% url 'polls:detail'
question.id %}" > Vote again?< /a >
{ % endblock % }
```

Step-6: Let's create the **navigation** bar for our application. Create a folder '**partials**' inside the folder 'templates' and then create a file '**_navbar.html**' inside the 'partial' folder. File structure will be templates->partials->_navbar.html. Write the code given below in this file.

Python

```
<nav class = "navbar navbar-dark bg-primary mb-4" >
   <div class = "container" >
        <a class = "navbar-brand" href = "/" > Pollster < /a
>
    </div >
</nav >
```

Step-7: We haven't included the head and body tag in every single HTML file we have created till now. We can write these codes in just one single file **base.html** and we can give the layout to our page. We will also bring our navigation bar(_navbar.html file) on this page. So open **base.html** file inside the 'template' folder and write down the code given below.

Python

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

<head >
    <link rel = "stylesheet" href =
"https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
        integrity = "sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin = "anonymous" >
    <title > Pollster { % block title % }{ % endblock % } < /title >
</head >

<body >
<!--NavBar-->
    { % include 'partials/_navbar.html'%}
    <div class = "container" >
       <div class = "row" >
           <div class = ".col-md-6 m-auto" >
               { % block content % }{ % endblock%}
            </div >
        </div >
    </div >
</body >

</html >
```

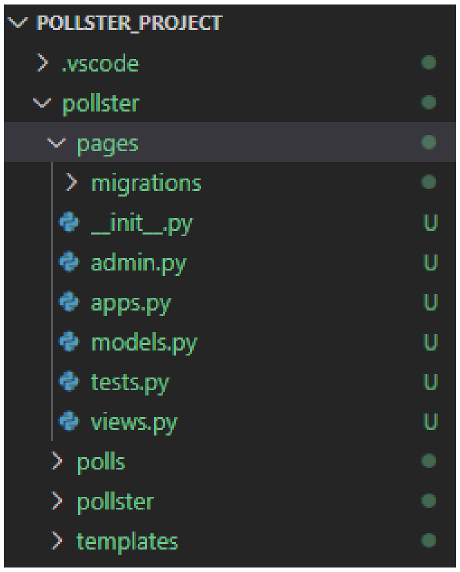
Create Landing Page

The URL **http://127.0.0.1:8000/** should display a landing page for our web application. So to create a landing page we will follow the step given below.

Step-1 Switch to the top-level pollster folder and run the command given below to create an app '**pages**'.

```
python manage.py startapp pages
```

Below is the folder structure once the 'pages' app will be created.



Step-2 Open '**views.py**' inside 'pages' folder i.e. pages->views.py. Write down the code given below to visit on landing page.

Python

```python
from django.shortcuts import render

# Create your views here.


def index(request):
    return render(request, 'pages / index.html')
```

Step-3 Create **urls.py** file inside the 'pages' folder i.e. pages->urls.py. Write the code given below to define the routing of pages->index.html file (check step-1).

Python

```python
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
        ]
```

Step-4 Create a folder '**pages**' inside 'template' folder. Now inside 'pages' folder create a file **index.html**. Write down the code given below to display the landing page to the users.

Python

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

<div class = "card text-center" >
   <div class = "card-body" >
        <h1 > Welcome To Pollster!< /h1 >
        <p > This is an Polling Web Application built with 
Django < /p >
        <a class = "btn btn-dark" href = "{% url 
'polls:index' %}" >
           View Available Polls < /a >
    </div >
</div >
{ % endblock % }
```

Create routing inside the main urls.py file of the application

We have created two apps in our application '**polls**' and '**pages**'. We need to define the routing of these two apps inside the main **urls.py** file which is pollster->pollster->urls.py file. So open the main **urls.py** file inside the pollster folder and write down the code given below to define the routing of these two apps('polls' and 'pages').

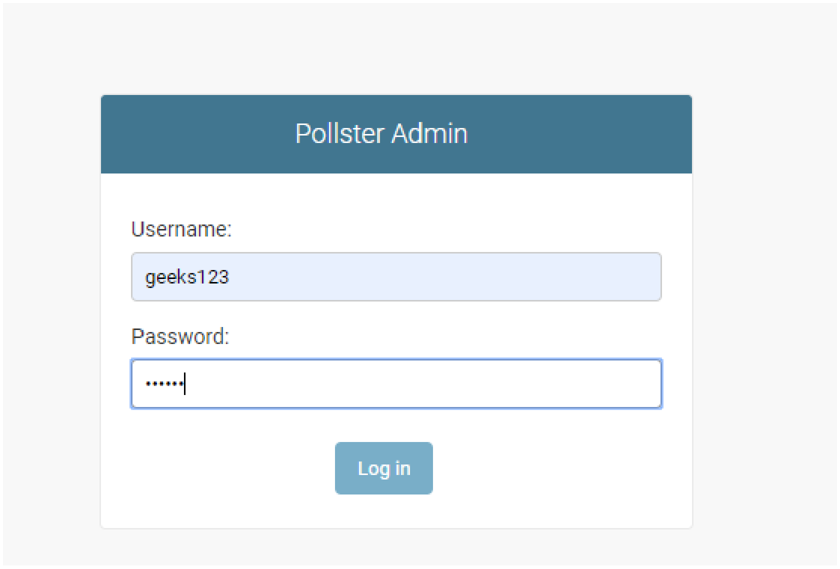
Python

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('', include('pages.urls')),
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

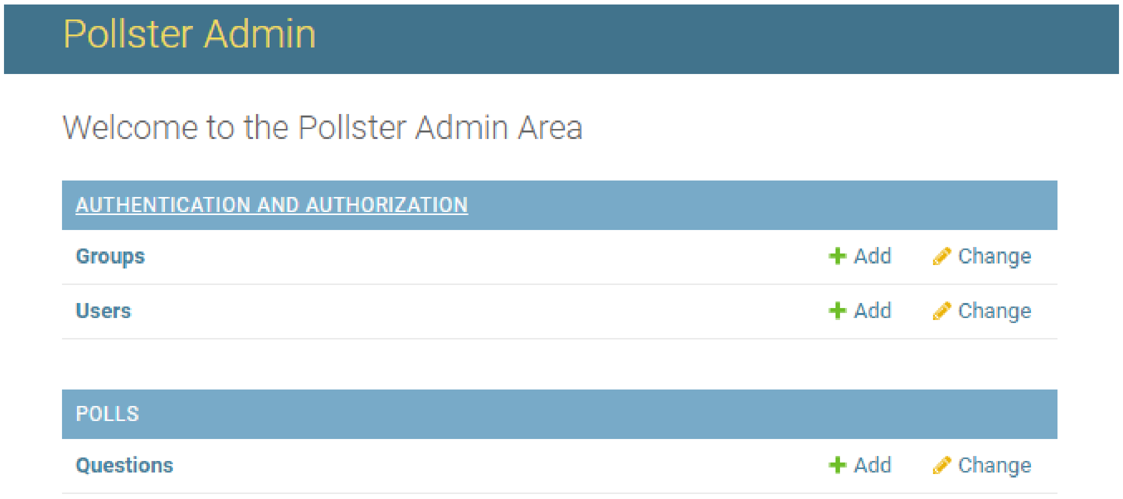
Testing of the Application

Admin Frontend

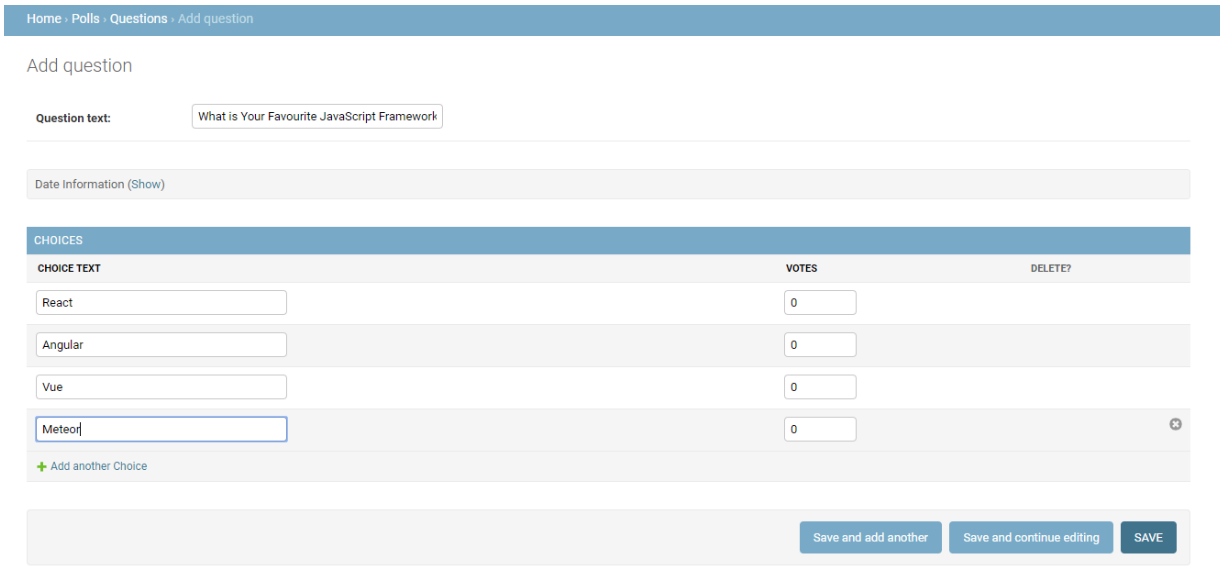
Step-1 Run the server using the command **python manage.py runserver** and browse the URL **http://127.0.0.1:8000/admin/**. Now enter the username and password to login into the system.



Step-2 Click on 'add' button next to the 'Questions'.



Step-2 Now add question and choices for those questions. Also, mention the date and time and then click on the 'save' button. You can add as many questions as you want. You will see a list of questions added in the database.

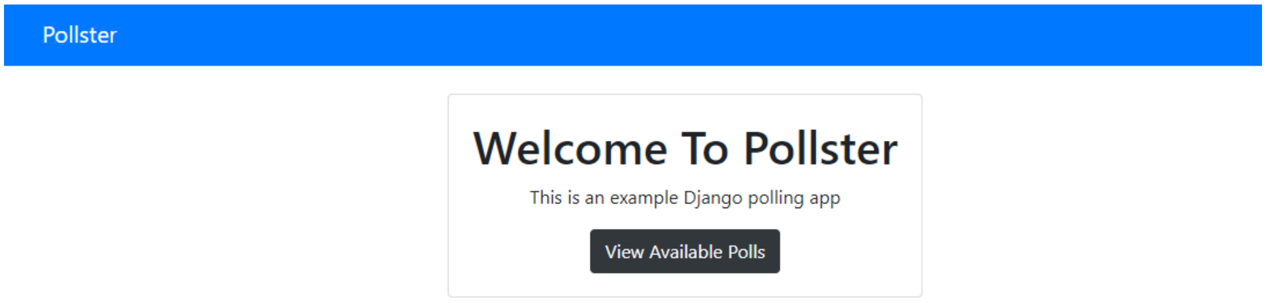


Question text: What is Your Favourite JavaScript Framework

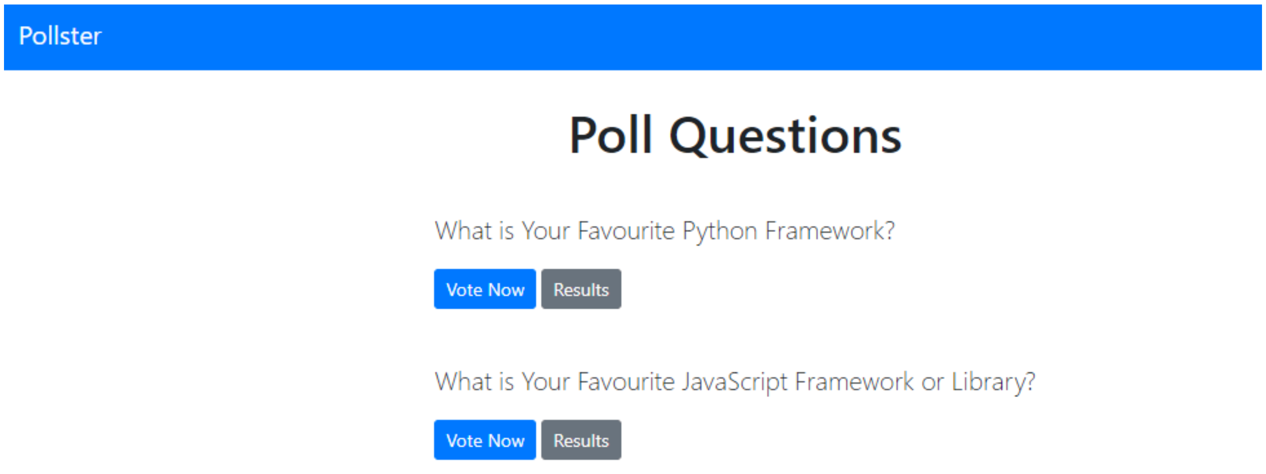
CHOICE TEXT	VOTES	DELETE?
React	0	
Angular	0	
Vue	0	
Meteor	0	

User Frontend

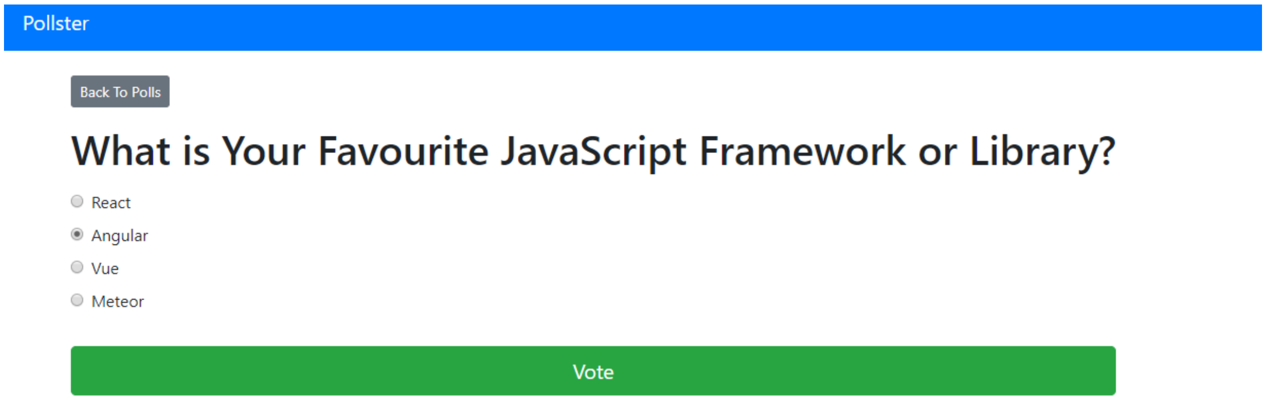
Step-1: Browse the URL **http://127.0.0.1:8000/** and you will see the landing page of the application. Click on the "View Available Polls"



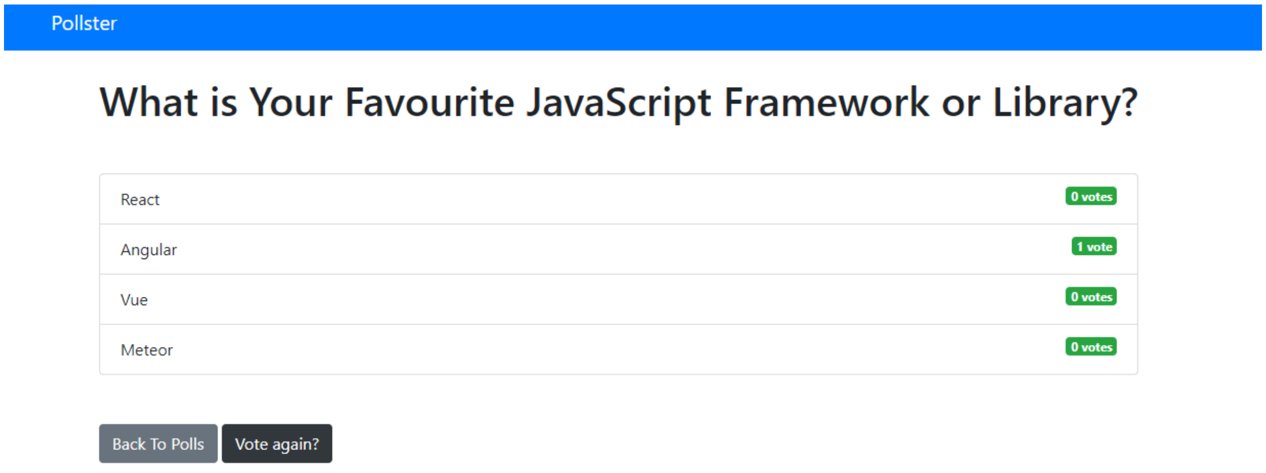
Step-2: You will see list of questions with two options 'Vote Now' and 'Results'. From here you need to select one question and click on the 'Vote Now' button.


Pollster
Poll Questions
What is Your Favourite Python Framework?
Vote Now
Results
What is Your Favourite JavaScript Framework or Library?
Vote Now
Results

Step-3: Once this is done select any one choice and click on 'Vote' button. You can also go to the previous menu using the 'Back to Polls' button on the top.



You will see the total voting result for the question you have selected.



React	0 votes
Angular	1 vote
Vue	0 votes
Meteor	0 votes

You can also check the total votes for any question using the option 'Results' from the 'Poll Questions' page.

Future Scope

This project can be used to conduct the online voting system in any field or industry. The project can be expanded and several other features can also be included based on the requirement. People can share the opinion and they can also check the total voting given by many users.

Project Repository Link

https://github.com/anuupadhyay/pollster-django-crash

anuu... 21

Previous Article

Django project to create a Comments System

Next Article

Determine The Face Tilt Using OpenCV - Python

Similar Reads

Integrating Django with Reactjs using Django REST Framework

In this article, we will learn the process of communicating between the Django Backend and React js frontend using the Django REST Framework. For the sake...

15+ min read

Project Idea - Online Voting Portal

As we know we all are going through this pandemic of COVID-19. We all are facing many problems in our day-to-day lives. Not only us but even all the...

4 min read

Joke Application Project Using Django Framework

Django is a high-level Python-based Web Framework that allows rapid development and clean, pragmatic design. It is also called batteries included...

2 min read

Top 10 Reasons to Choose Django Framework For Your Project

When it comes to choosing a new language or framework for a project what matters to most of the developers? Simplicity? Reliability? Built-in packages to...

9 min read

Adding Tags Using Django-Taggit in Django Project

Django-Taggit is a Django application which is used to add tags to blogs, articles etc. It makes very easy for us to make adding the tags functionality to our djang...

2 min read

Create a Voting App using React-Native

Voting involves assessing multiple entities based on specific criteria. This article guides the creation of a basic voting app, where users can compare and evaluat...

3 min read

College Management System using Django - Python Project

In this article, we are going to build College Management System using Django and will be using dbsqlite database. In the times of covid, when education has...

15+ min read

What is Decentralized Voting Application (DApps)?

The Project Name is Decentralized Voting Application (DApps) which is built on Solidity Language. This Project showcases a lot of Solidity's features. It...

4 min read

Implement Token Authentication using Django REST Framework

Token authentication refers to exchanging username and password for a token that will be used in all subsequent requests so to identify the user on the server...

2 min read

How to Create a basic API using Django Rest Framework ?

Django REST Framework is a wrapper over the default Django Framework, basically used to create APIs of various kinds. There are three stages before...

4 min read

Article Tags : GBlog Project Python Django-Projects +1 More

Practice Tags : python

In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

DSA

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial

Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science

Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

DevOps

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Inteview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles