Voting System Project Using Django Framework

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 give voting 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.

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.

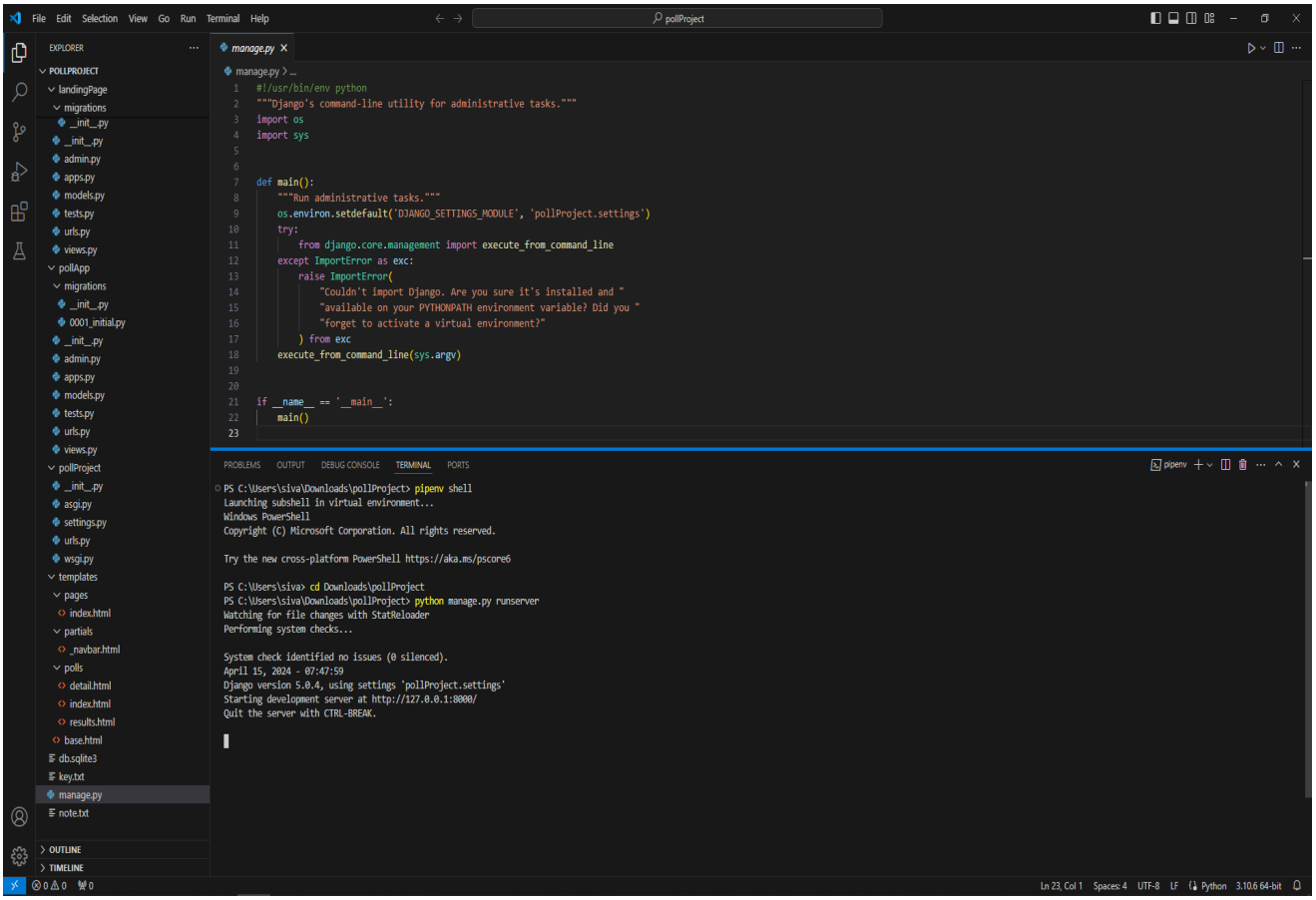
Technologies used in the project: Django framework and SQLite database which comes by default with Django.

Implementation of the Project

Creating Project

Step-1: Create an empty folder **pollster_project** in your directory.
Step-2: Now switch to your folder and create a virtual environment in this folder using the following command.

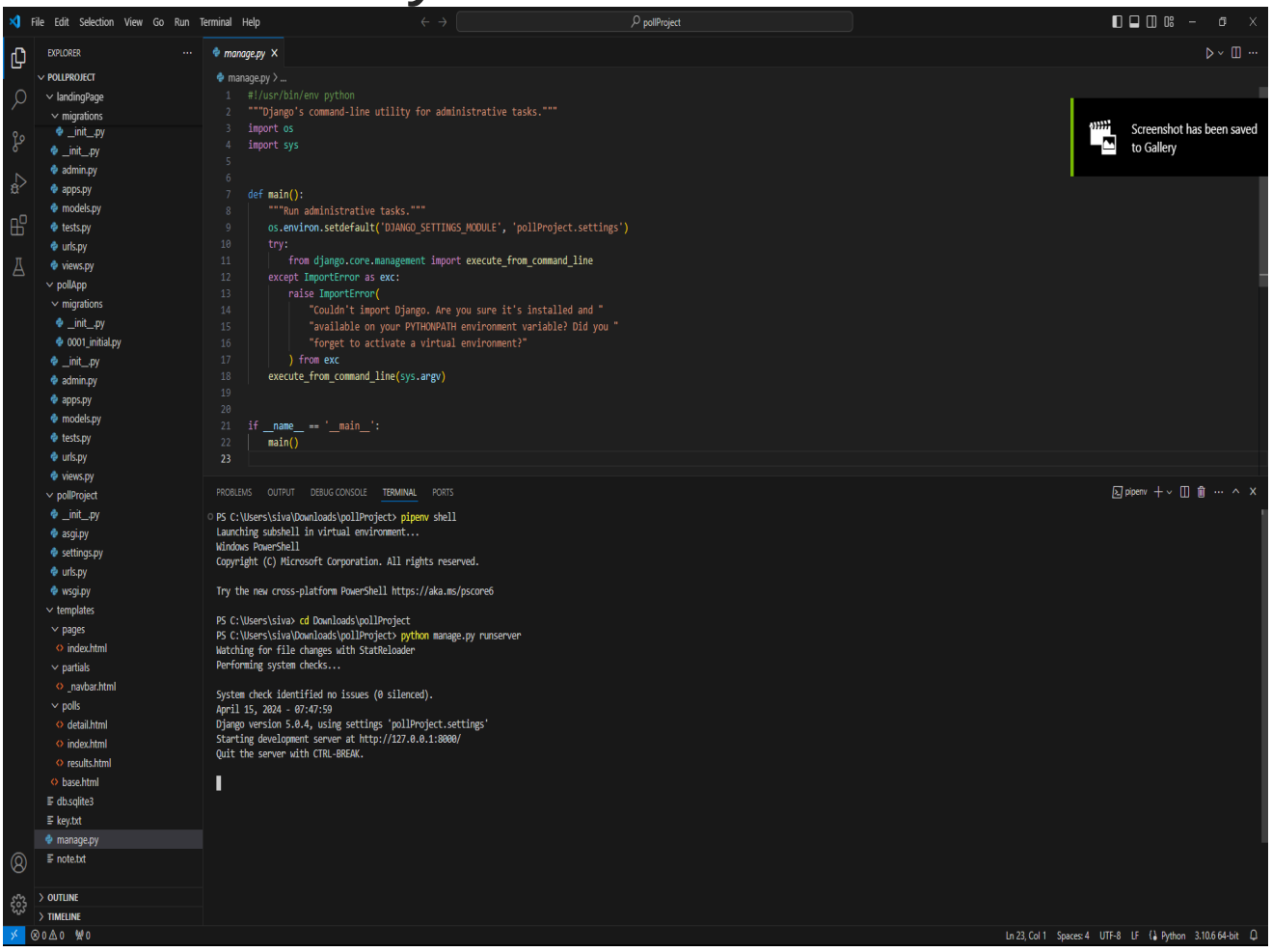
pip install pipenv
pipenv shell



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

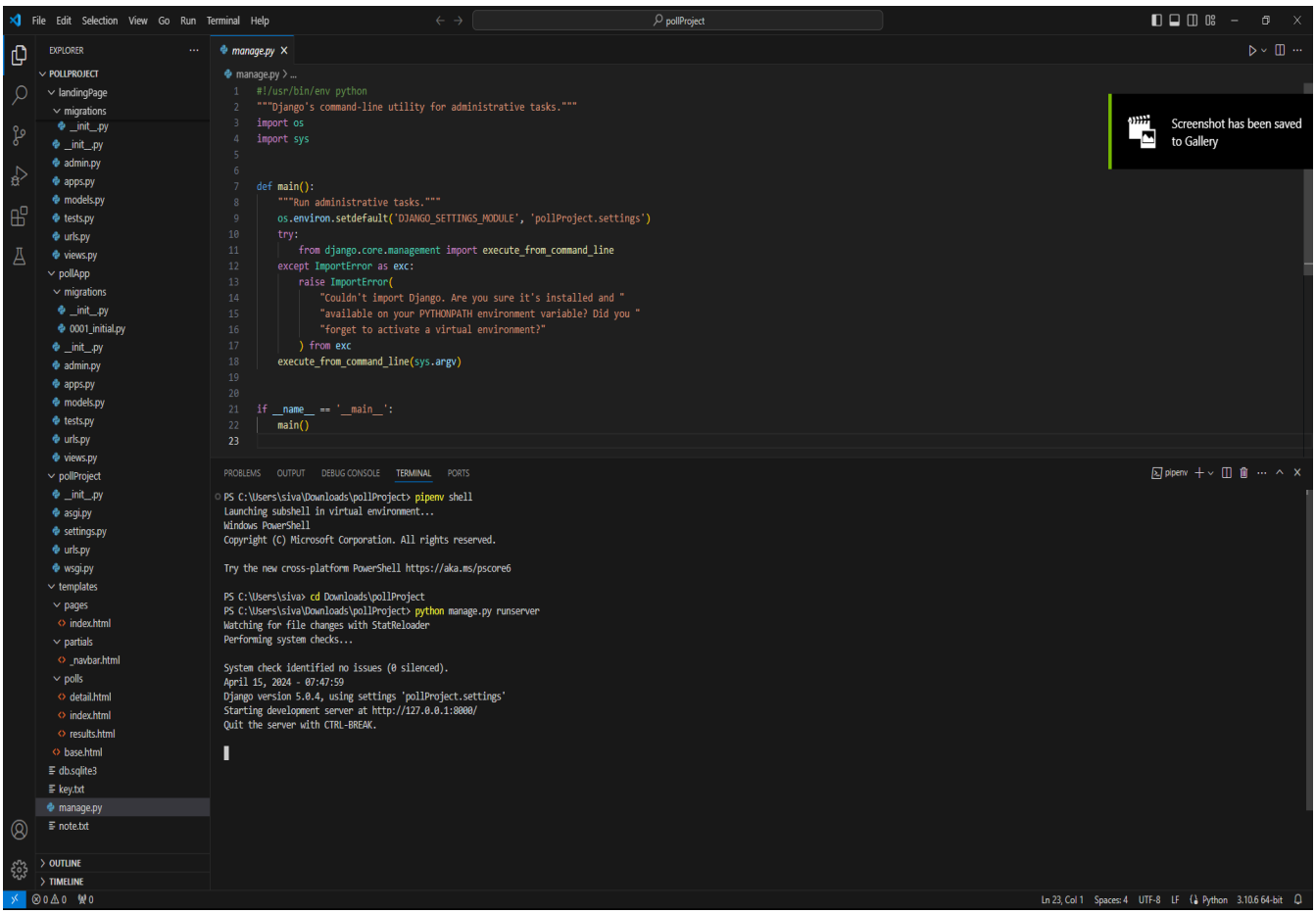
check if the application running or not using your **http://127.0.0.1:8000/** in your browser.

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.



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 Code:

```
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):
```

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.

Python3

```
INSTALLED_APPS = [

    'polls.apps.PollsConfig',

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',
```

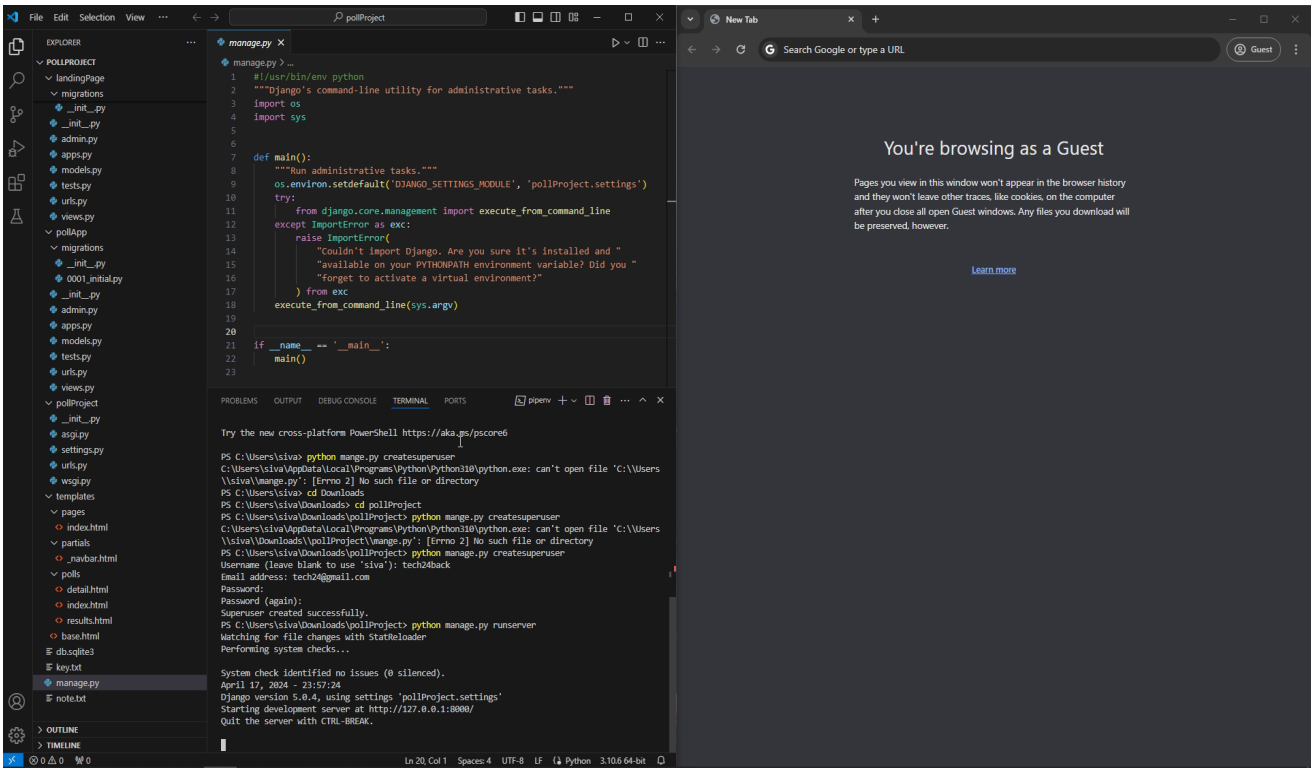
```
    'django.contrib.staticfiles',

]
```

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

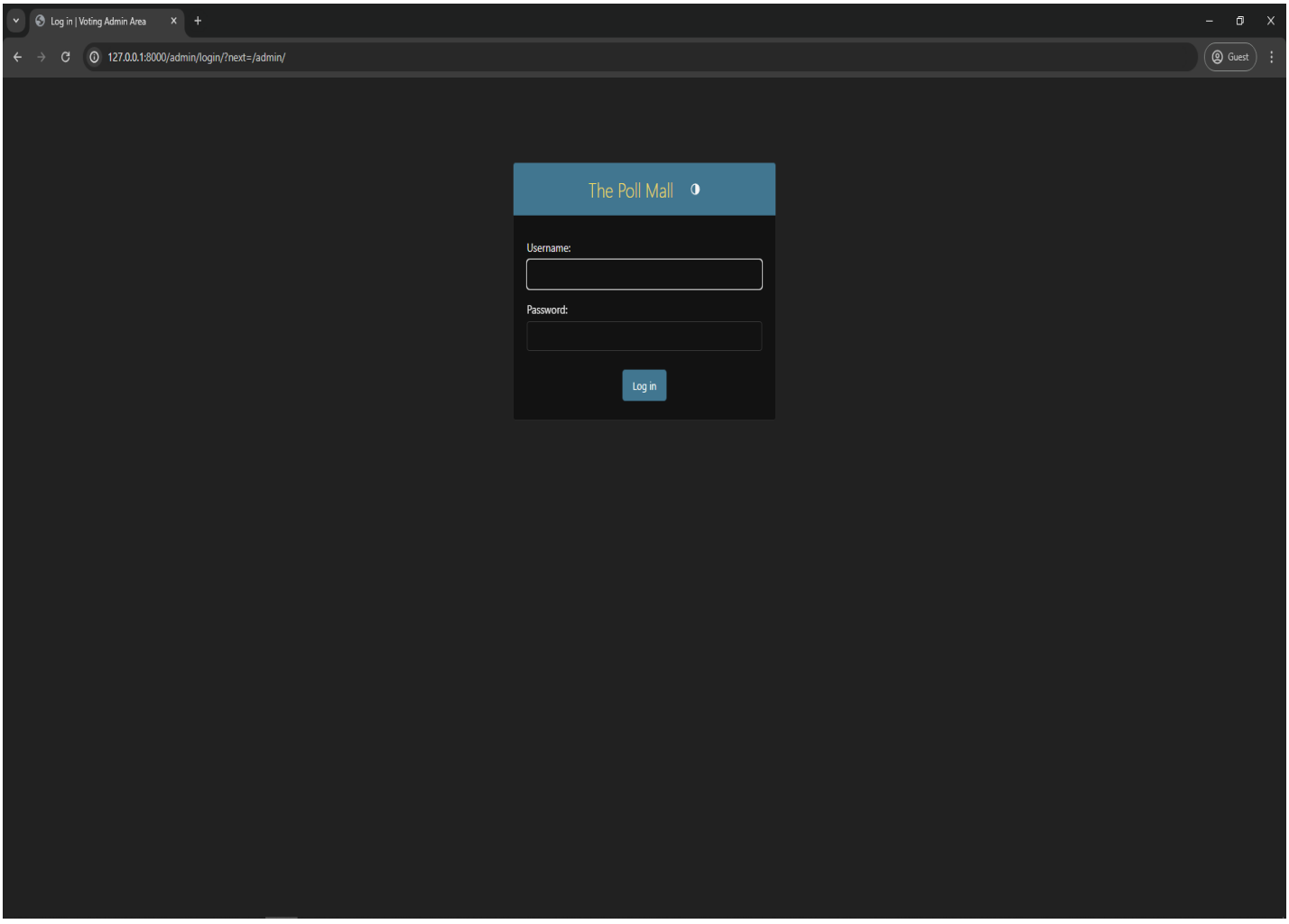
It will prompt username which we need to enter.

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.

Python3

```
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"
```

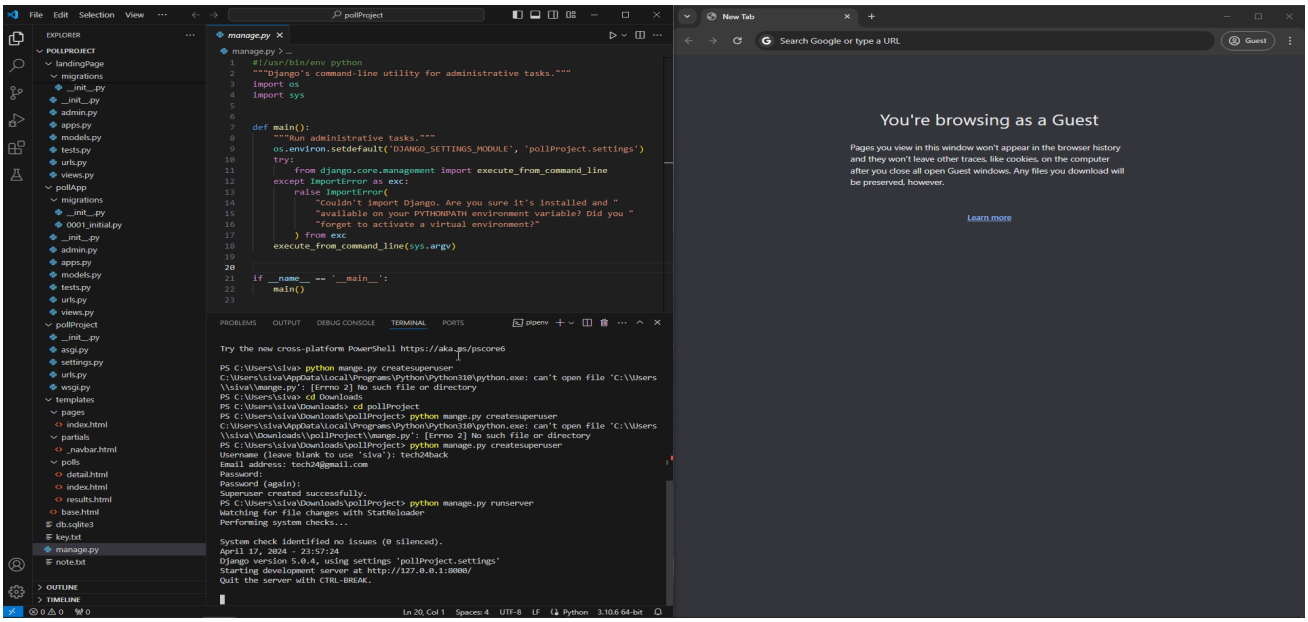
```
admin.site.index_title = "Welcome to the
Pollster Admin Area"




class ChoiceInLine(admin.TabularInline):

    model = Choice

    extra = 3




class
QuestionAdmin(admin.ModelAdmin):

    fieldsets = [(None, {'fields':
```

['question_text']}), ('Date Information', {

'fields': ['pub_date'], 'classes': ['collapse']}),]

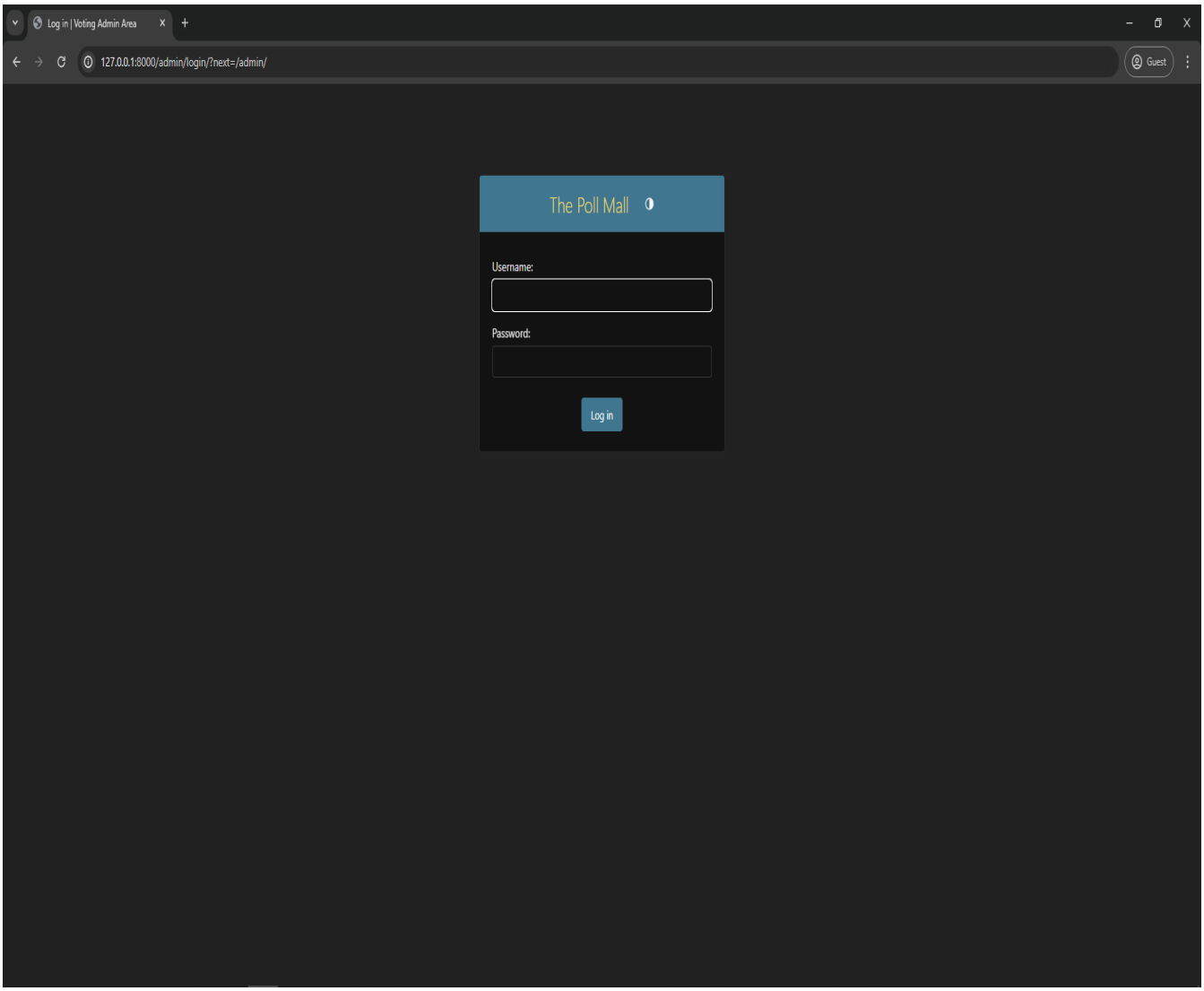
inlines = [ChoiceInLine]

admin.site.register(Question, QuestionAdmin)



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.

Python3

```
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…

Python3

```
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.

Python3

```
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_process
```

```
ors.request',

                'django.contrib.auth.context_pr
ocessors.auth',

                'django.contrib.messages.cont
ext_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).

Python3

```
{% 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 %}

<pNo polls available</p
```

```
{% endif %}

{% endblock %}


<html lang="en"



    <link                              rel="stylesheet"
href="https://stackpath.bootstrapcdn.com
/bootstrap/4.4.1/css/bootstrap.min.css"

        integrity="sha384-
Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6u
g5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifj
h" crossorigin="anonymous"
```

```
    <titlePollster {% block title %}{% endblock %}</title

</head




    <!--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.

Python3

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).

Python3

```
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.

Python3

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

{% block content %}
```

```
<div class="card text-center"

    <div class="card-body"

        <h1Welcome To Pollster!</h1

        <pThis 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’).

Python3

```
from django.contrib import admin

from django.urls import include, path
```

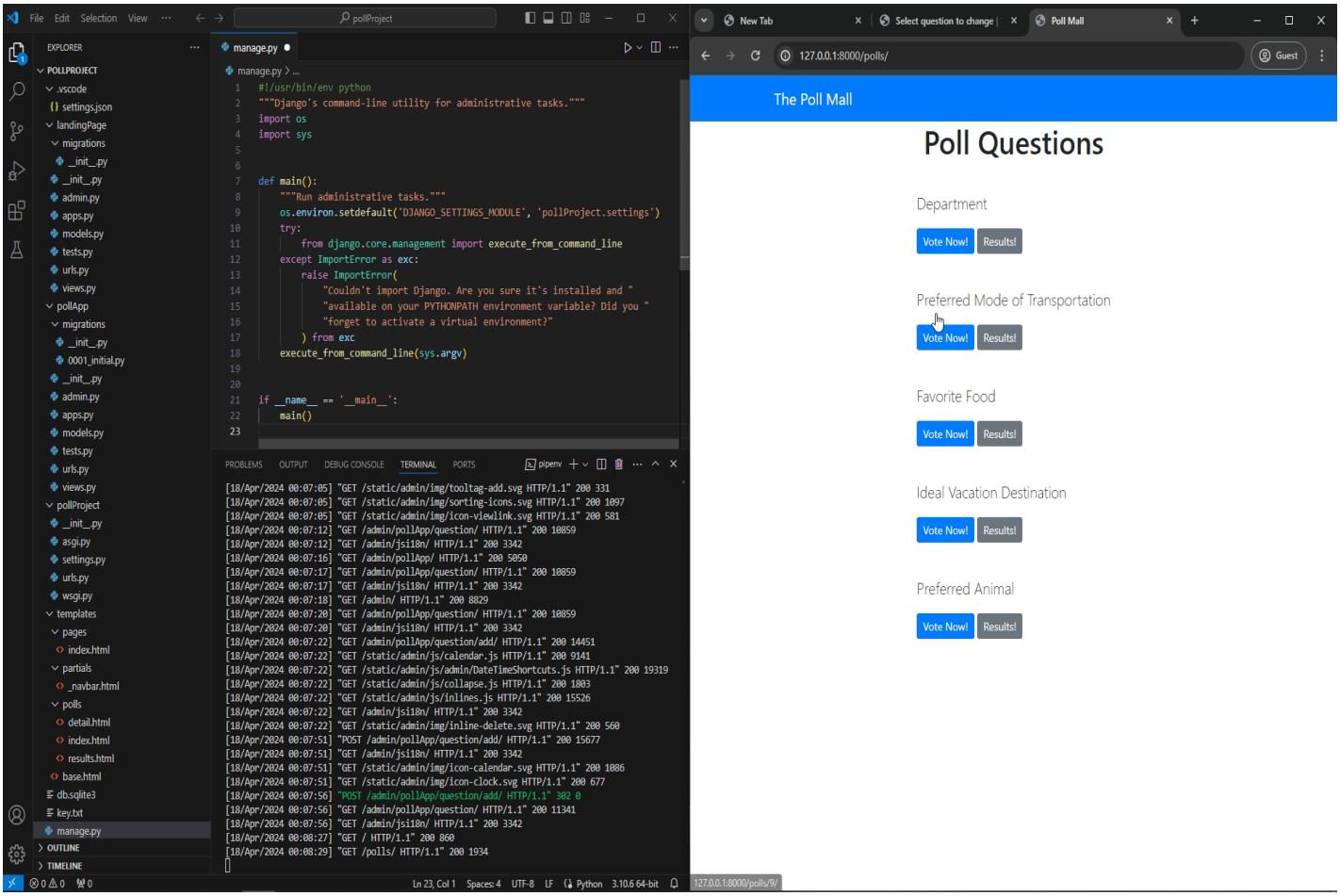
```
urlpatterns = [
    path('', include('pages.urls')),
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

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’.

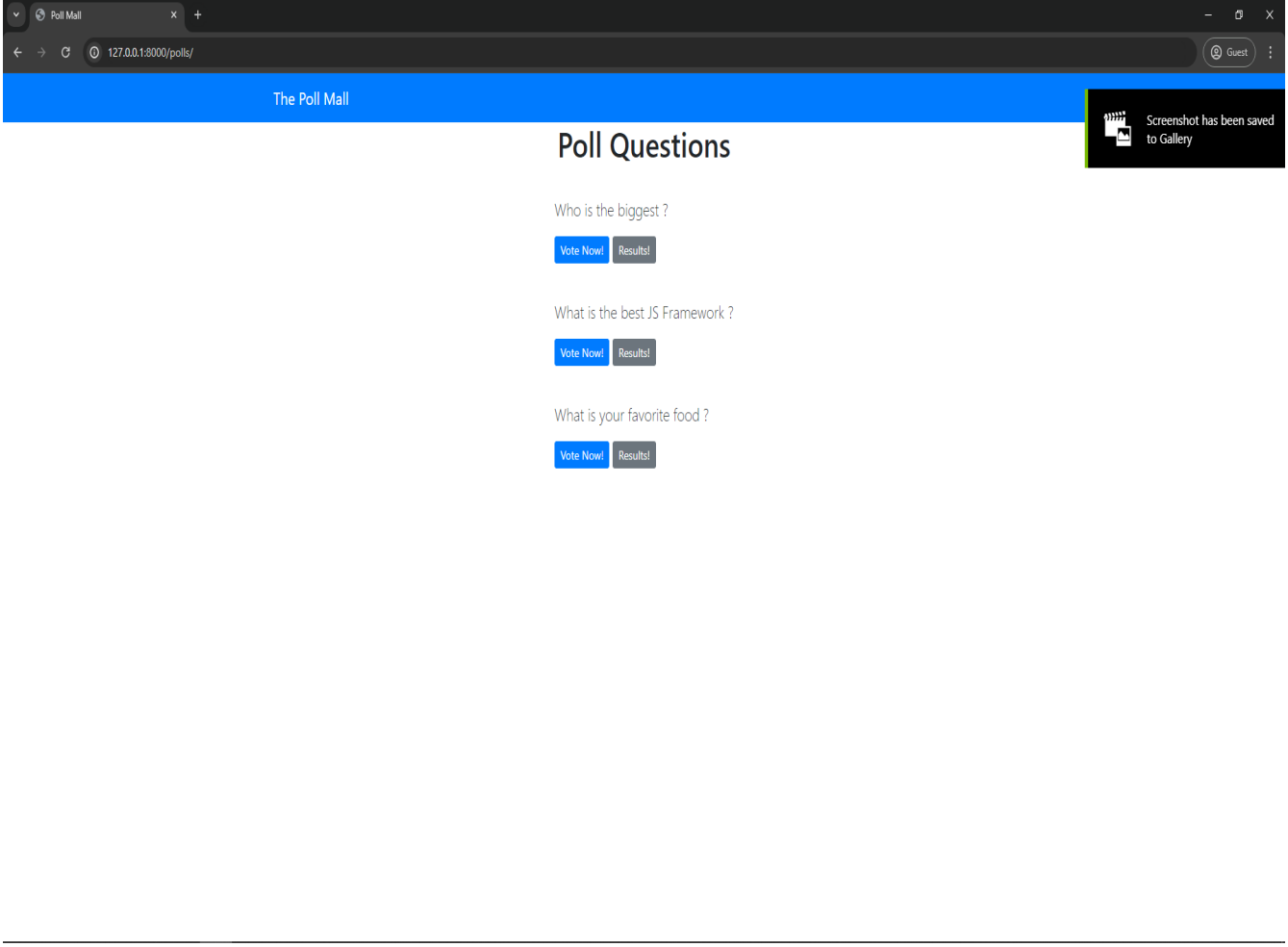
Step-2 Now add question and choices for those questions. Also, mention the date and time and then click on the ‘save’ button. You can add as many

questions as you want. You will see a list of questions added in the database.

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

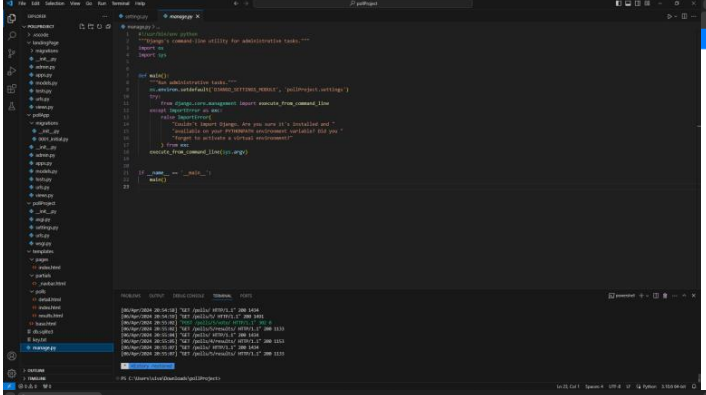

Poll Mall
127.0.0.1:8000/polls/
Guest
The Poll Mall
Screenshot has been saved to Gallery
Poll Questions
Who is the biggest ?
Vote Now!
Results!
What is the best JS Framework ?
Vote Now!
Results!
What is your favorite food ?
Vote Now!
Results!

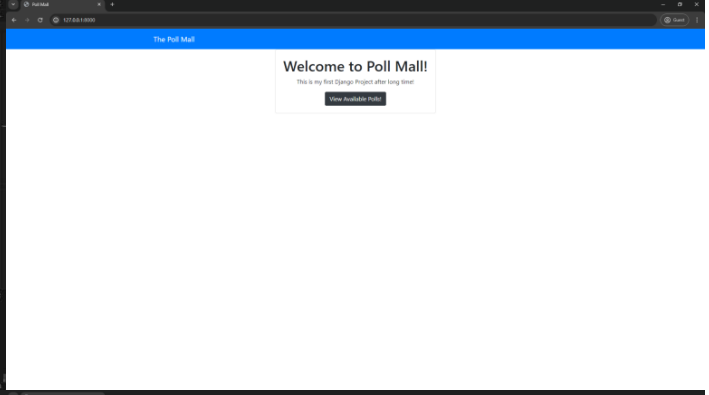
two options ‘Vote Now’ and ‘Results’. From here you need to select one question and click on the ‘Vote Now’ button.

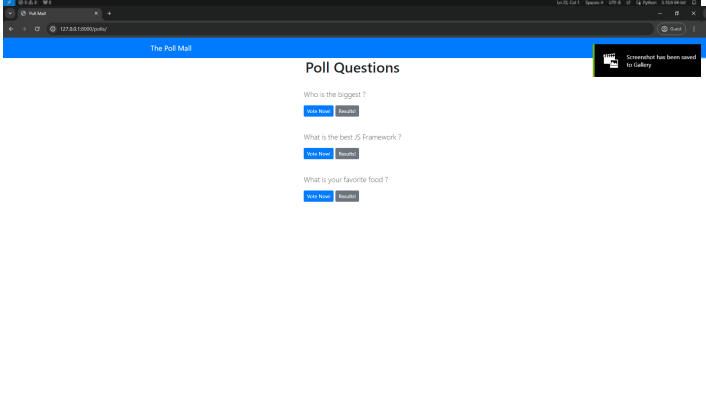
Step-3: Once this is done select any one choice and click on ‘Vote’ button. You can also go to the previous menu using the ‘Back to Polls’ button on the top.

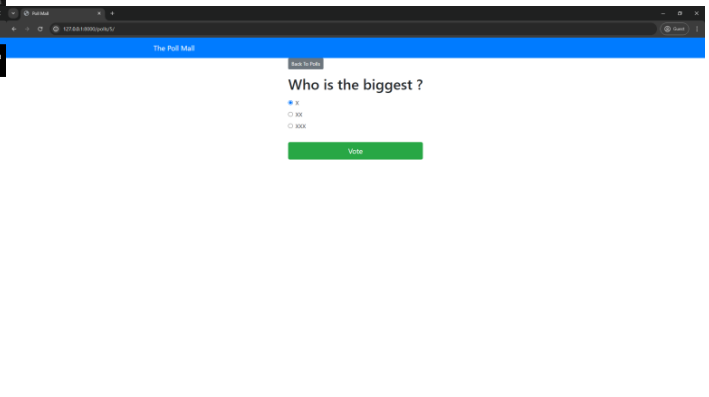
You will see the total voting result for the question you have selected.

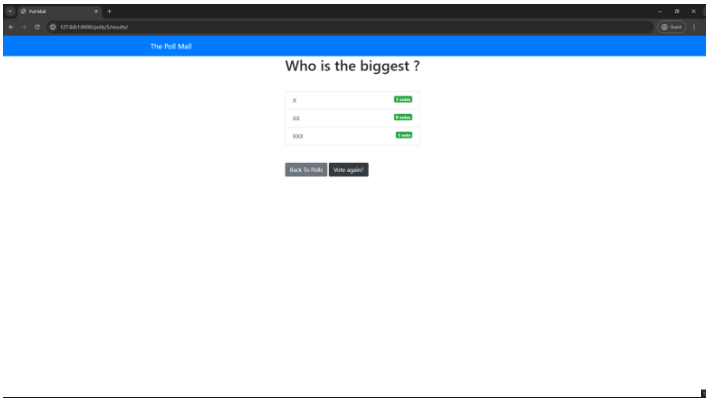
You can also check the total votes for any question using the option ‘Results’ from the ‘Poll Questions’ page.

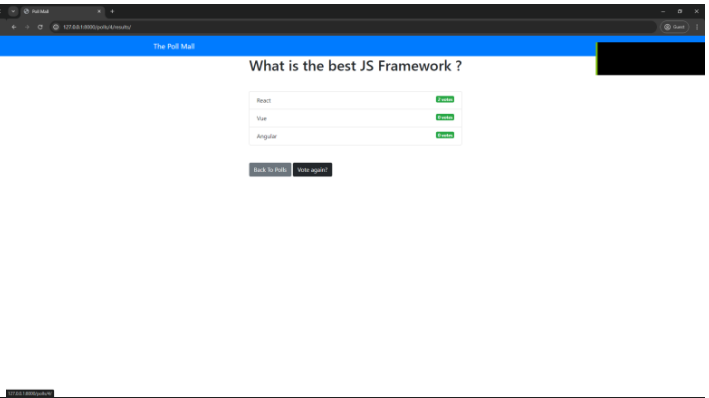


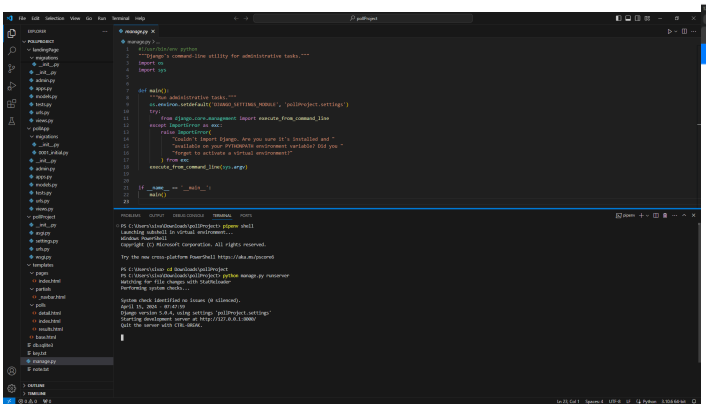

The Poll Mall
Welcome to Poll Mall!
This is my first Django Project after long time!
View Available Polls!

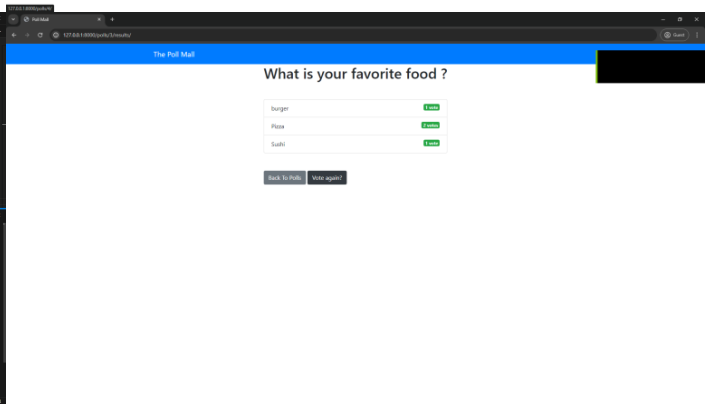

The Poll Mall
Poll Questions
Who is the biggest ?
Vote Now!
Results
What is the best JS Framework ?
Vote Now!
Results
What is your favorite food ?
Vote Now!
Results
Screenshot has been saved to Gallery

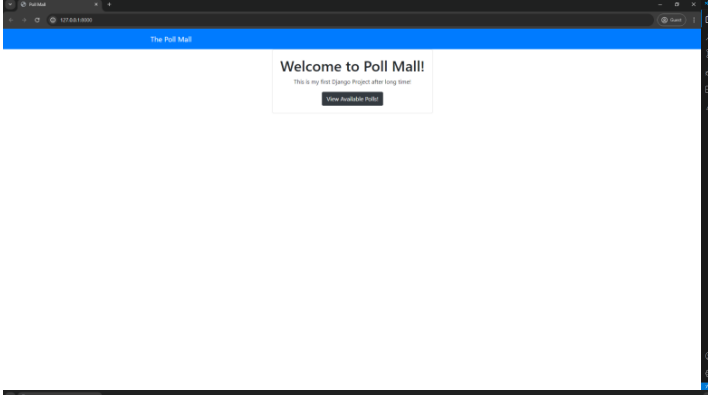

The Poll Mall
Back To Polls
Who is the biggest ?
x
xx
xxx
Vote

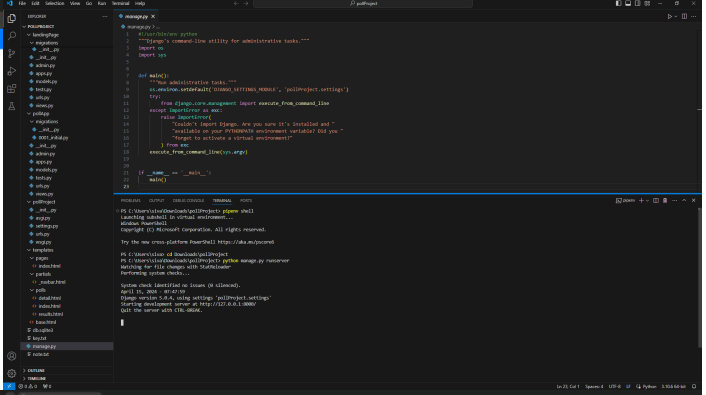

The Poll Mall
Who is the biggest ?
x
2 votes
xx
2 votes
xxx
1 vote
Back To Polls
Vote again?

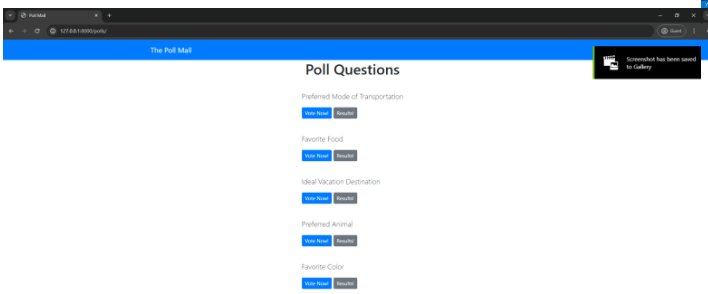

The Poll Mall
What is the best JS Framework ?
React
2 votes
Vue
0 votes
Angular
0 votes
Back To Polls
Vote again?

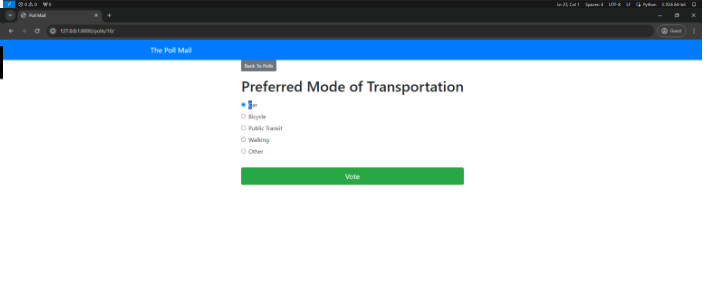

The Poll Mall
Poll Questions
Preferred Mode of Transportation
Vote Now!
Results
Favorite Food
Vote Now!
Results
Ideal Vacation Destination
Vote Now!
Results
Preferred Animal
Vote Now!
Results
Favorite Color
Vote Now!
Results
Screenshot has been saved to Gallery

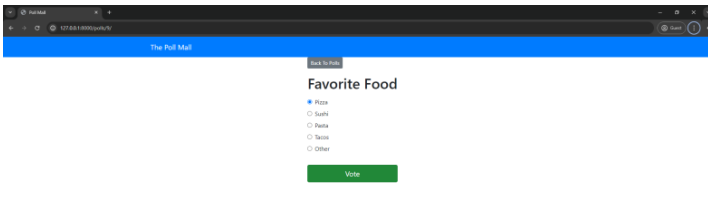

The Poll Mall
What is your favorite food ?
burger
1 vote
Pizza
2 votes
Sushi
1 vote
Back To Polls
Vote again?

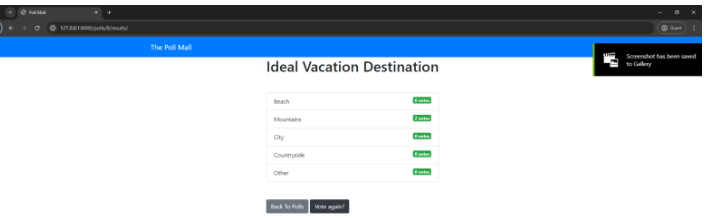

The Poll Mall
Welcome to Poll Mall!
This is my first Django Project after long time!
View Available Polls!

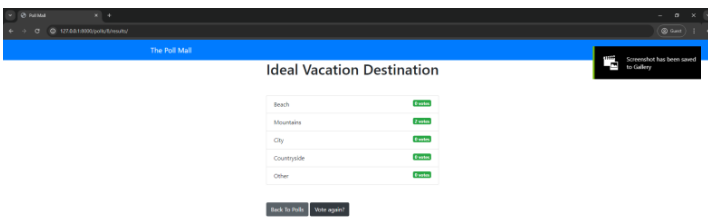


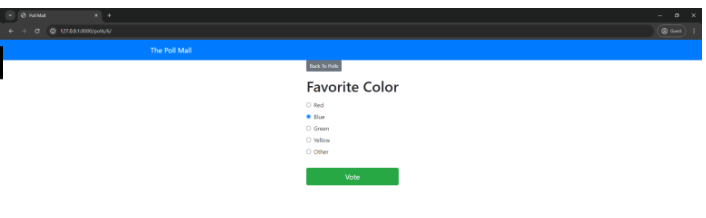

The Poll Mall
Poll Questions
Preferred Mode of Transportation
Vote Now
Results
Favorite Food
Ideal Vacation Destination
Preferred Animal
Favorite Color
Screenshot has been saved to Gallery

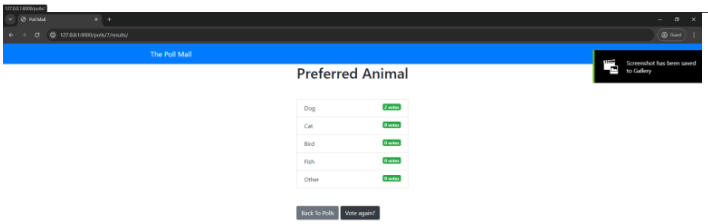

The Poll Mall
Back To Polls
Preferred Mode of Transportation
Car
Bicycle
Public Transit
Walking
Other
Vote

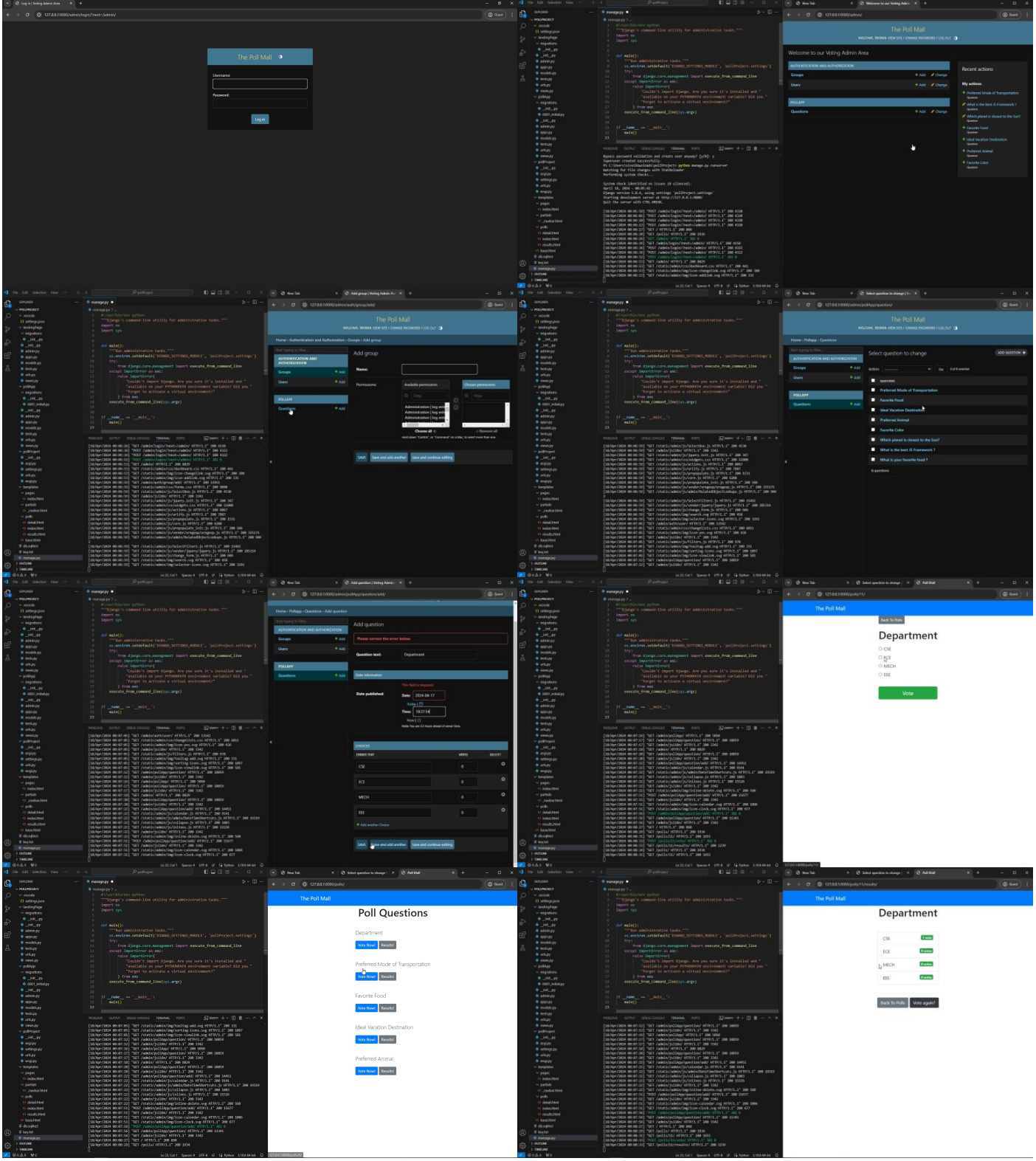

The Poll Mall
Back To Polls
Favorite Food
Pizza
Sushi
Pasta
Tacos
Other
Vote


The Poll Mall
Ideal Vacation Destination
Beach
Mountains
City
Countryside
Other
Back To Polls
Vote again?
Screenshot has been saved to Gallery


The Poll Mall
Ideal Vacation Destination
Beach
Mountains
City
Countryside
Other
Back To Polls
Vote again?
Screenshot has been saved to Gallery


The Poll Mall
Back To Polls
Favorite Color
Red
Blue
Green
Yellow
Other
Vote


The Poll Mall
Preferred Animal
Dog
Cat
Bird
Fish
Other
Back To Polls
Vote again?
Screenshot has been saved to Gallery



Conclusion:

A voting application built with the Django framework could be a dynamic platform for conducting polls, surveys, or elections online. Users could register, create, and participate in various voting events. Features might include user authentication, multiple-choice or ranked voting options, real-time result tracking, and admin controls for managing polls and user data. Django's built-in security measures and flexibility make it well-suited for ensuring the integrity and reliability of such applications.