**Django poll application report**

Project steps:

We are going to go through a step-by-step process of implementing this project.

- Firstly, we are going to create the directory for our project, which we will call djangotutorial, using the following command in the terminal:

**...\>** **mkdir** djangotutorial

- Then, run the following command to bootstrap a new Django project:

**...\>** django-admin startproject mysite djangotutorial

- Our project directory should look like this:

djangotutorial/

manage.py

mysite/

\_\_init\_\_.py

settings.py

urls.py

asgi.py

wsgi.py

- To make sure our project works properly, change into the djangotutorial directory and run the following command:

**...\>** py manage.py runserver

- You should see the following output after you run the command above:

Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.

Run 'python manage.py migrate' to apply them.

August 29, 2025 - 15:50:53

Django version 5.2, using settings 'mysite.settings'

Starting development server at <http://127.0.0.1:8000/>

Quit the server with CONTROL-C.

WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead.

For more information on production servers see: <https://docs.djangoproject.com/en/>5.2/howto/deployment/

- Now we are going to create our polls app inside the djangotutorial folder and type this command:

**...\>** py manage.py startapp polls

- The previous command will create a polls directory looking like this:

polls/

\_\_init\_\_.py

admin.py

apps.py

migrations/

\_\_init\_\_.py

models.py

tests.py

views.py

- Next is to set up our database, which consists of several steps, and firstly, we type this command:

**...\>** py manage.py migrate

cmm

- Then we create our models, which resemble our database table layout. We will create two models, which are **Question** and **Choice,** by typing the following code in the **polls/models.py** file:

import datetime

from django.db import models

from django.utils import timezone

class Question(models.Model):

    question\_text = models.CharField(max\_length=200)

    pub\_date = models.DateTimeField("date published")

    def \_\_str\_\_(self):

        return self.question\_text

    def was\_published\_recently(self):

now = timezone.now()

        return now - datetime.timedelta(days=1) <= self.pub\_date <= now

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

- To include the app in our project, we need to add a reference to its configuration class in the **INSTALLED\_APPS** setting. Edit the **mysite/settings.py** file and add that dotted path to the **INSTALLED\_APPS** setting. It’ll look like this:

INSTALLED\_APPS = [

"polls.apps.PollsConfig",

"django.contrib.admin",

"django.contrib.auth",

"django.contrib.contenttypes",

"django.contrib.sessions",

"django.contrib.messages",

"django.contrib.staticfiles",

]

- Now Django knows to include the **polls** app. Let’s run another command:

**...\>** py manage.py makemigrations polls

- After running the above command, you should see the following:

Migrations for 'polls':

polls/migrations/0001\_initial.py

+ Create model Question

+ Create model Choice

- Then type in this command that will run the migrations for you and manage your database schema automatically:

**...\>** py manage.py sqlmigrate polls 0001

- You should see something similar to the following after running the above command:

BEGIN;

*--*

*-- Create model Question*

*--*

CREATE TABLE "polls\_question" (

"id" bigint NOT NULL PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,

"question\_text" varchar(200) NOT NULL,

"pub\_date" timestamp with time zone NOT NULL

);

*--*

*-- Create model Choice*

*--*

CREATE TABLE "polls\_choice" (

"id" bigint NOT NULL PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,

"choice\_text" varchar(200) NOT NULL,

"votes" integer NOT NULL,

"question\_id" bigint NOT NULL

);

ALTER TABLE "polls\_choice"

ADD CONSTRAINT "polls\_choice\_question\_id\_c5b4b260\_fk\_polls\_question\_id"

FOREIGN KEY ("question\_id")

REFERENCES "polls\_question" ("id")

DEFERRABLE INITIALLY DEFERRED;

CREATE INDEX "polls\_choice\_question\_id\_c5b4b260" ON "polls\_choice" ("question\_id");

COMMIT;

- Now we run the migrate command again to create those model tables in our database:

**...\>** py manage.py migrate

Operations to perform:

Apply all migrations: admin, auth, contenttypes, polls, sessions

Running migrations:

Rendering model states... DONE

Applying polls.0001\_initial... OK

- To create an admin user to log in to the admin site, we type this command:

**...\>** py manage.py createsuperuser

- Firstly, we enter the username:

Username: admin

- Then, we enter our email address:

Email address: admin@example.com

- Lastly, we enter the password:

Password: \*\*\*\*\*\*\*\*\*\*

Password (again): \*\*\*\*\*\*\*\*\*

Superuser created successfully.

- To make the polls app visible on the admin site we should edit the **polls/admin.py** file as below:

from django.contrib import admin

from .models import Question

admin.site.register(Question)

- Now we are going to add our views to the **polls/views.py** file, which will work as the logic for each page. We have four views:

1. IndexView, which is a ListView that displays the latest 5 published questions.

2. DetailView, which shows a specific question chosen by the user and its choices.

3. ResultView, is a DetailView that displays the results or number of votes for each choice for a specific question.

4. Vote, which handles the voting logic and shows an error if no choice is selected.

- We need to connect these new views to the polls.urls module by adding the following code to the **polls/urls.py** file:

from django.urls import path

from . import views

app\_name = "polls"

urlpatterns = [

    path("", views.IndexView.as\_view(), name="index"),

    path("<int:pk>/", views.DetailView.as\_view(), name="detail"),

    path("<int:pk>/results/", views.ResultsView.as\_view(), name="results"),

    path("<int:question\_id>/vote/", views.vote, name="vote"),

]

- Next step is to create a directory called templates in the polls directory and inside the templates directory, we create another directory called polls, which will contain the .html files, so we will add three .html files, so the path should look like this: **polls/templates/polls/index.html.**

**1. Index.html:**

{% load static %}

<link rel="stylesheet" href="{% static 'polls/style.css' %}">

{% if latest\_question\_list %}

    <ul>

    {% for question in latest\_question\_list %}

        <li><a href="{% url 'polls:detail' question.id %}">{{ question.question\_text }}</a></li>

    {% endfor %}

    </ul>

{% else %}

    <p>No polls are available.</p>

{% endif %}

**2. detail.html**

<form action="{% url 'polls:vote' question.id %}" method="post">

{% csrf\_token %}

<fieldset>

    <legend><h1>{{ question.question\_text }}</h1></legend>

    {% if error\_message %}<p><strong>{{ error\_message }}</strong></p>{% endif %}

    {% for choice in question.choice\_set.all %}

        <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}">

        <label for="choice{{ forloop.counter }}">{{ choice.choice\_text }}</label><br>

    {% endfor %}

</fieldset>

<input type="submit" value="Vote">

</form>

**resu**

**3. results.html**

<h1>{{ question.question\_text }}</h1>

<ul>

{% for choice in question.choice\_set.all %}

    <li>{{ choice.choice\_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>

{% endfor %}

</ul>

<a href="{% url 'polls:detail' question.id %}">Vote again?</a>

- Lastly, we are going to add code for testing in the **polls/test.py** file. We will test whether questions with a publication date in the future or the past are displayed in the detail view, and verify that if there are no questions, an appropriate message is displayed.

What I Learned

As someone who initially had no prior knowledge of Django, this training has allowed me to gain a solid understanding of both Django and web development in general.

One of the first key concepts I learned was the difference between the **MVC (Model–View–Controller)** pattern and Django’s **MVT (Model–View–Template)** architecture. In Django, the *template* corresponds to the view in MVC, while the *view* in Django serves the role of the controller by handling the application’s logic.

I also became familiar with the overall Django project structure, its layered architecture, and how the main components, models, views, templates, and URLs interact with each other to build a web application.

Additionally, I gained practical experience with Django’s database system. By defining models such as *Question* and *Choice* in models.py, I learned how to represent data in the application. I also learned how to apply migrations to create and update database tables, which gave me a deeper understanding of how Django manages data.

Another main thing I have learned about is the concept of views, as it serves the role of the controller that handles the page's logic and templates that are responsible for the presentation of data, where the templates consist of HTML files.

I have also learned about mapping URLs to the views using urls.py, such as mapping the results URL (<http://127.0.0.1:8000/polls/1/results/>) to the ResultsView, and learned how URL patterns connect browser requests to view logic. Understood how to use the admin interface to add questions or choices and edit them.

I learned how to write unit tests for the polls app and came to understand the importance of testing for ensuring reliability and correctness, while also becoming more aware of Django’s philosophy of “Don’t Repeat Yourself” (DRY) and the value of building reusable apps.

This is a brief conclusion of what I have learned throughout my training period, and working on this simple yet important project has given me a solid foundation in Django.