



---

# NEXT GEN EMPLOYABILITY PROGRAM

---

## CAPSTONE PROJECT SHOWCASE

### PROJECT TITLE

---

**Voting Application using Django Framework-  
DHANANJAYAN.B (4019,AEC)**

---

Abstract | Problem Statement | Project Overview | Proposed Solution |  
Technology Used | Modelling & Results | Conclusion

## **Abstract**

The proposed voting application is a web-based platform that allows users to create and participate in online votes. The application is built using the Django framework, a popular and well-supported Python-based web framework that provides a robust foundation for building scalable and secure web applications . The application is also designed to be flexible and scalable, with a modular architecture that allows for easy customization and extension. This makes it suitable for a wide range of use cases, from small-scale internal votes to large-scale public elections. Overall, the proposed voting application is a secure, user-friendly, and flexible platform for conducting online votes. Its use of the Django framework ensures a robust and scalable foundation, while its focus on security and user experience makes it an ideal choice for a wide range of voting scenarios.

## **Problem Statement**

Online voting has become increasingly popular in recent years, with a growing number of organizations and governments turning to digital platforms to conduct elections and polls. However, online voting also presents a number of challenges, particularly in terms of security and integrity .

Overall, the proposed voting application will address the challenges of security and integrity in online voting, while also providing a user-friendly platform for conducting online votes. Its use of the Django

framework will ensure a robust and scalable foundation, while its focus on security and user experience will make it an ideal choice for a wide range of voting scenarios.

In addition to its focus on security,the application will also prioritize user experience,with a clean and intuitive interface that makes it easy for users to create and participate in votes.

## Project Overview

The project overview for a voting application using the Django framework involves creating a secure and user-friendly online voting system. The application allows users to register, vote, and view real-time results. Here are the steps involved in building the voting application:

- 1. Setting up a Django Project :** Create a Django project to serve as the foundation for the voting application.
- 2. Designing the Database Schema:** Define the database structure to store user information, votes, and other relevant data.
- 3. Creating User Authentication:** Implement user authentication to allow users to register, log in, and participate in voting.
- 4. Building the Voting Interface:** Develop the interface where users can view options, select their choices, and submit votes.
- 5. Implementing Real-time Results:** Display the voting results dynamically to provide instant feedback to users.
- 6. Developing an Admin Panel:** Build an admin panel to manage the voting process, candidates, and user accounts effectively.

## **Proposed Solution**

The proposed solution for a voting application using the Django framework is to create a secure and user-friendly online voting platform. The application will allow users to register, vote, and view real-time results. To build the application, the Django framework will be used as the foundation due to its robustness and scalability. The application will have a user-friendly interface, a secure database, real-time results, and an admin panel for efficient management of elections, candidates, and user accounts.

# Home Page



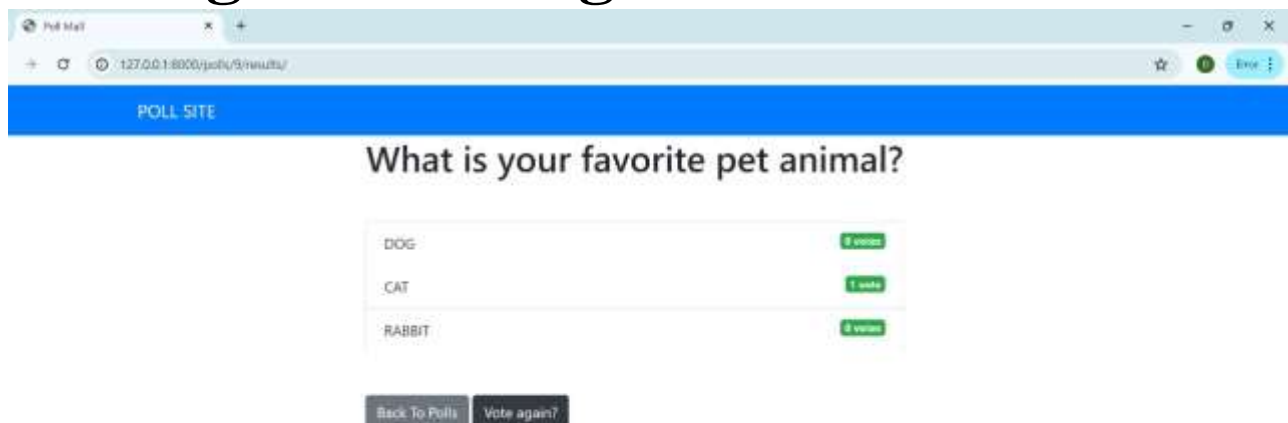
# Poll Page



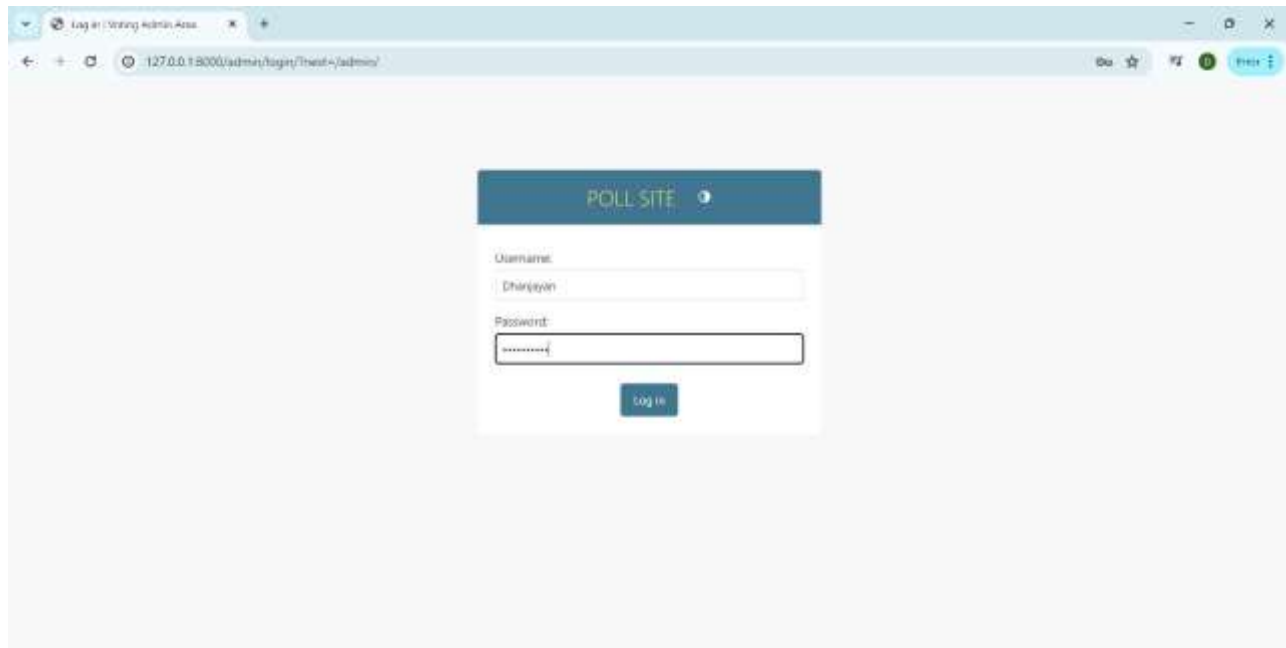
# Voting Page



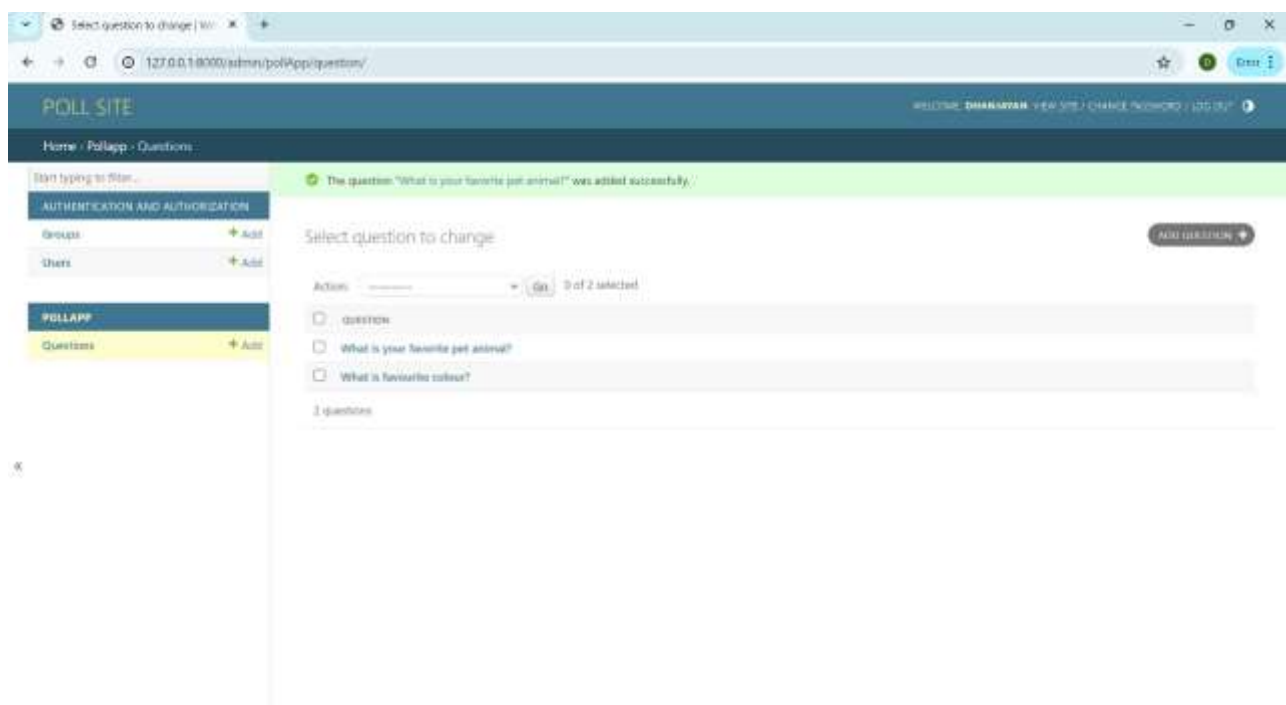
# Voting Details Page



# Admin Login Page

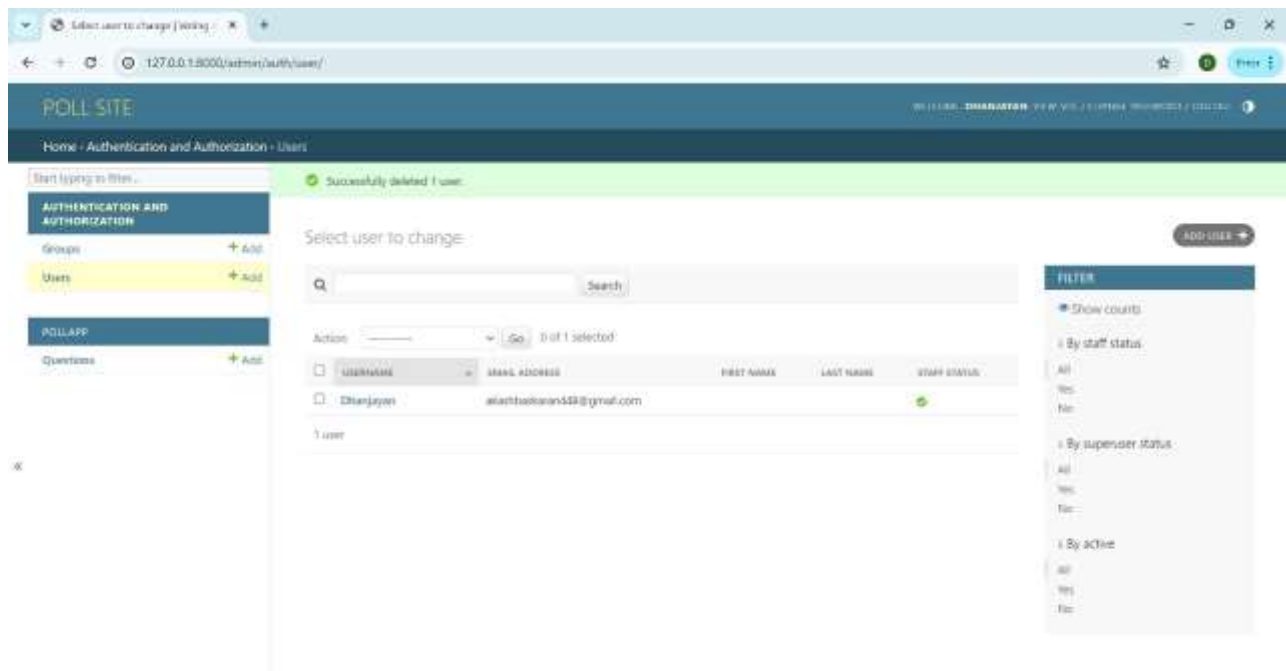


# Admin Home Page

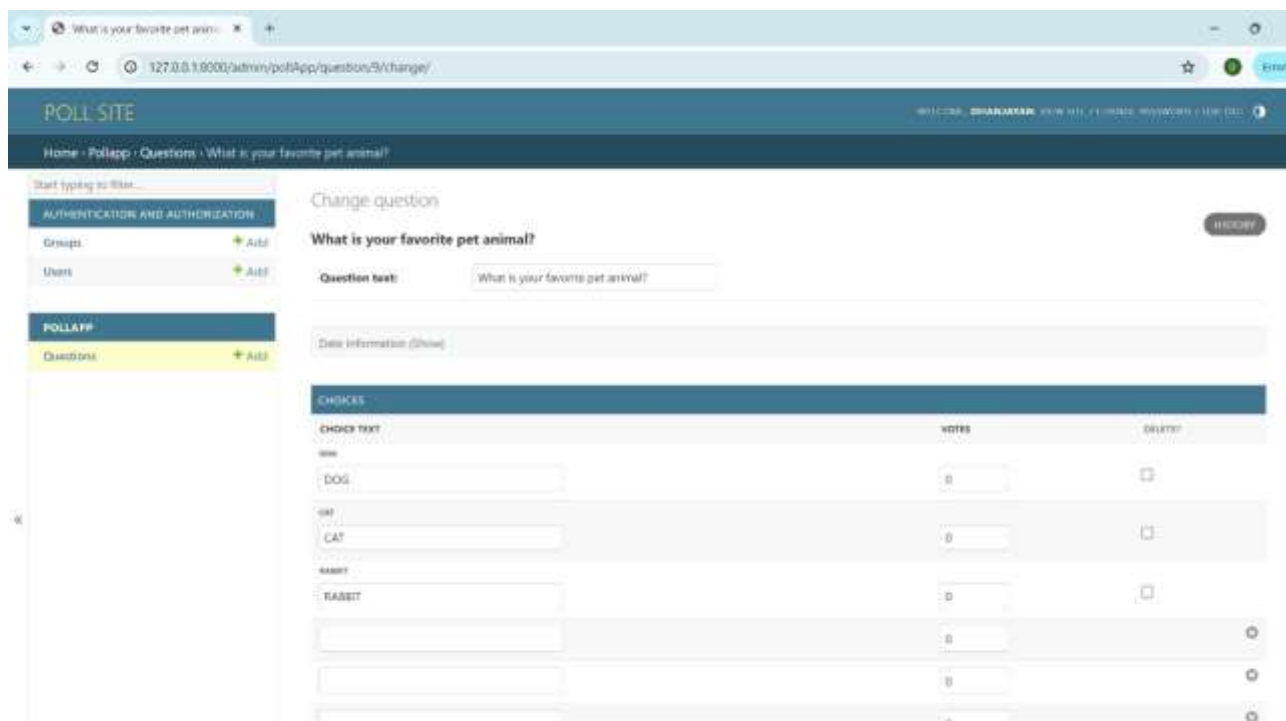




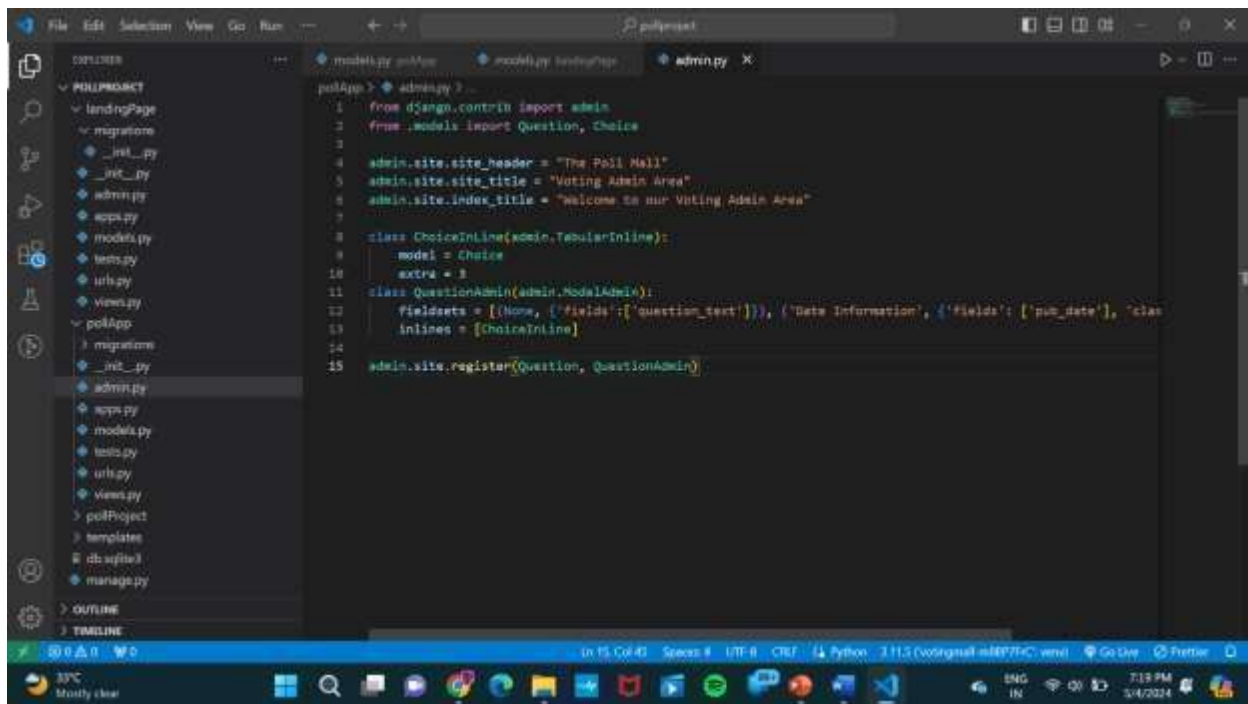
# Authentication and Authorization Page



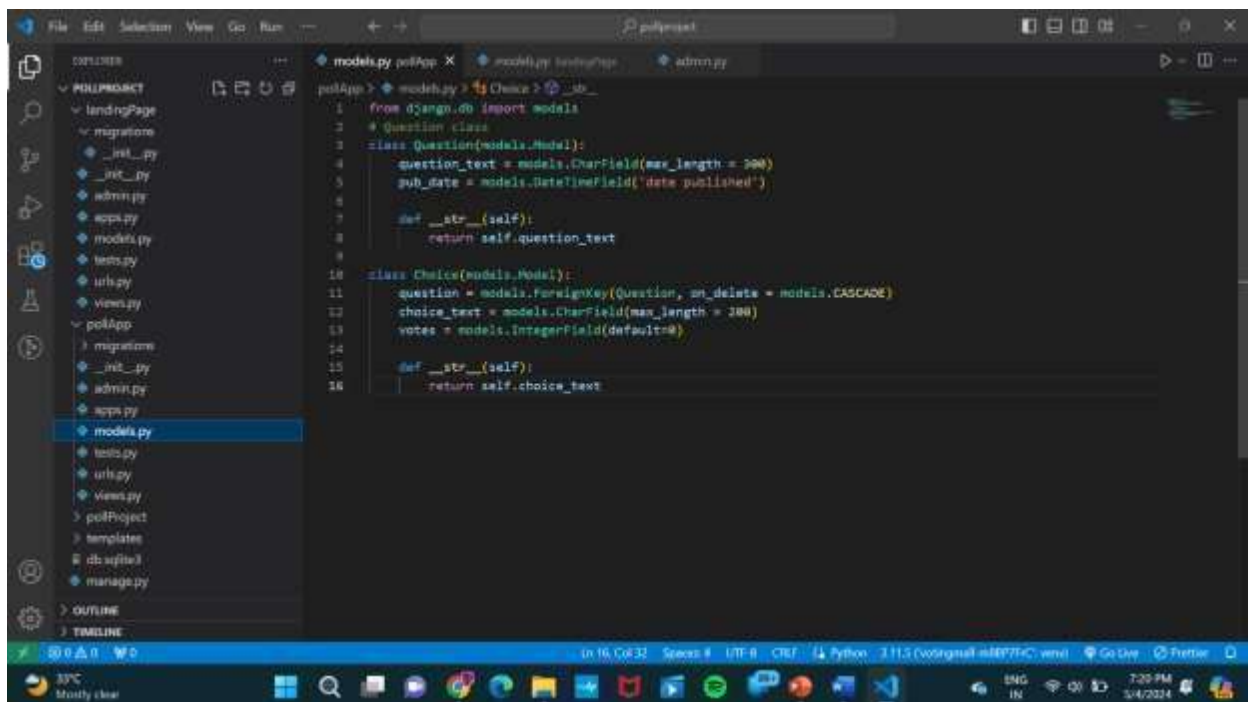
# Questions Adding Section Page



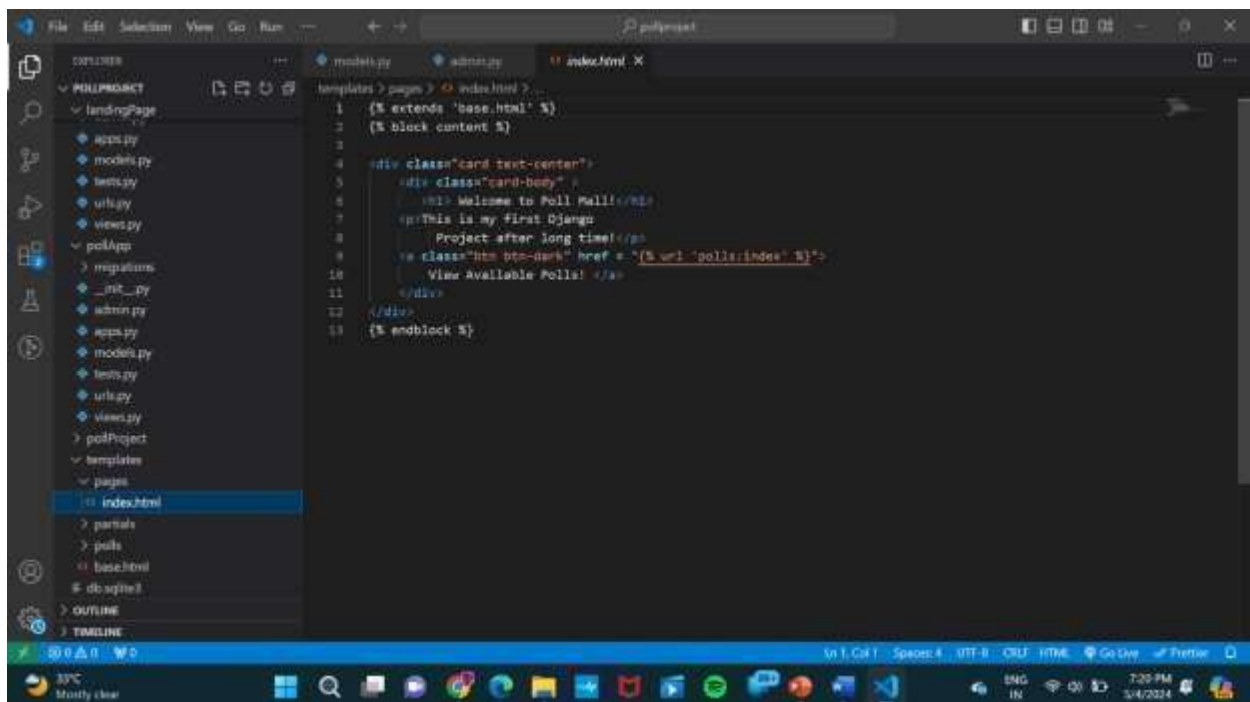
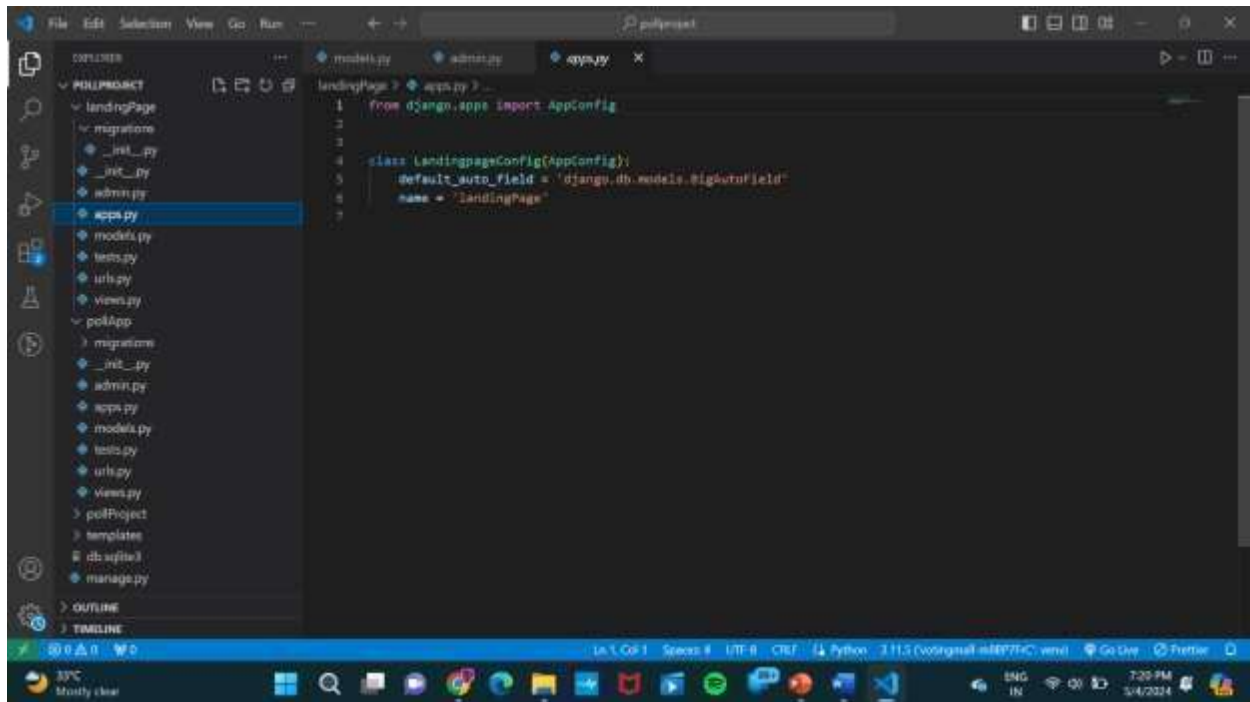
# IMPLEMENTATION

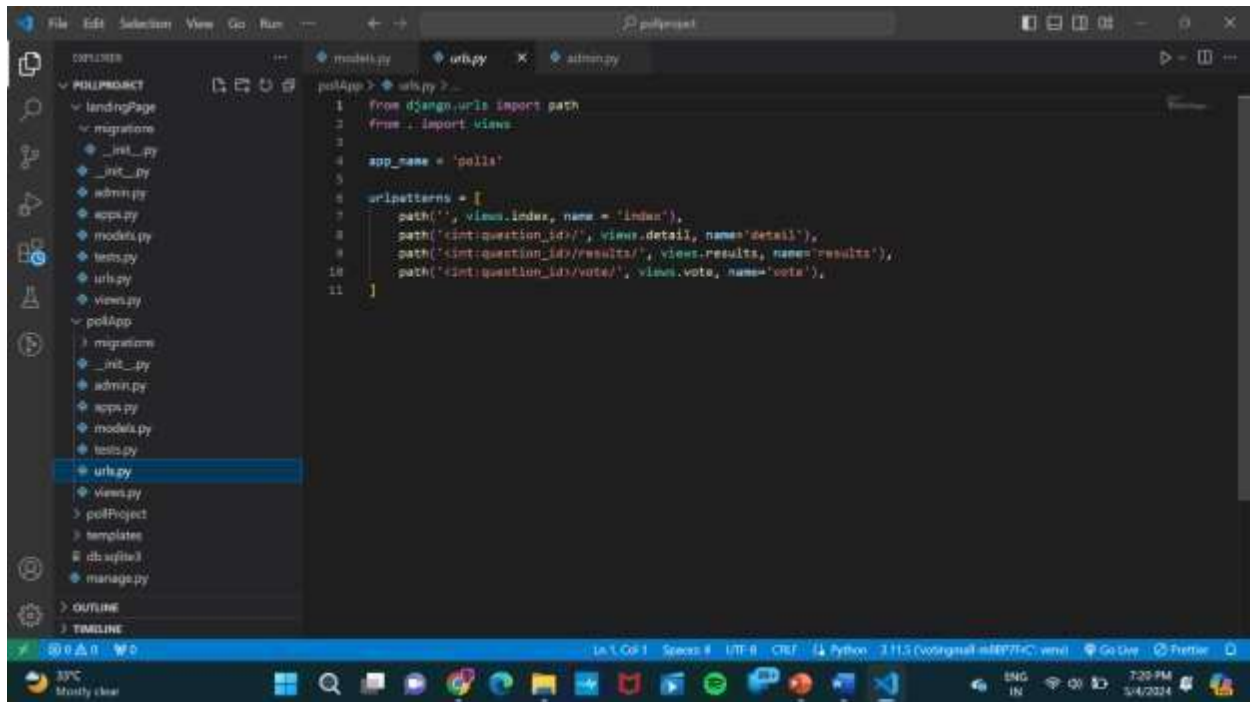


```
1 from django.contrib import admin
2 from .models import Question, Choice
3
4 admin.site.site_header = "The Poll Mall"
5 admin.site.site_title = "Voting Admin Area"
6 admin.site.index_title = "Welcome to our Voting Admin Area"
7
8 class ChoiceInline(admin.TabularInline):
9     model = Choice
10     extra = 3
11
12 class QuestionAdmin(admin.ModelAdmin):
13     fieldsets = [(None, {'fields': ('question_text',)}), ('Date Information', {'fields': ('pub_date',), 'class': 'collapse'})]
14     inlines = [ChoiceInline]
15
16 admin.site.register(Question, QuestionAdmin)
```



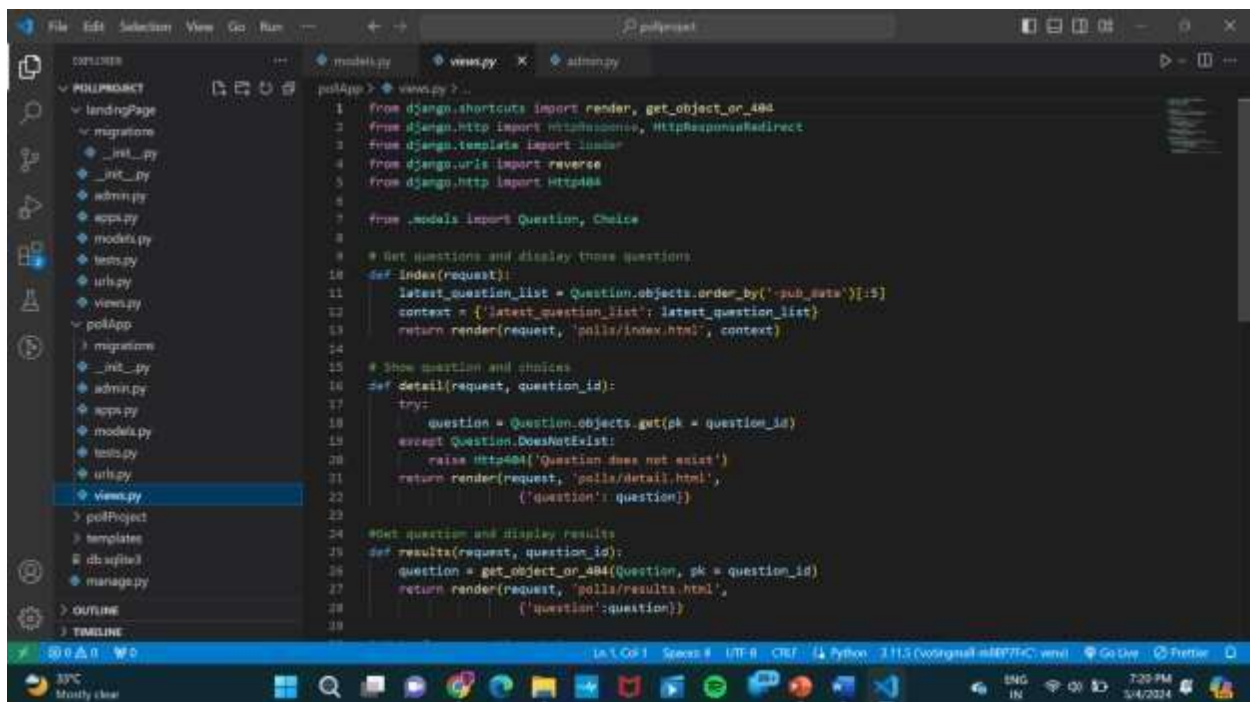
```
1 from django.db import models
2
3 class Question(models.Model):
4     question_text = models.CharField(max_length=200)
5     pub_date = models.DateTimeField('date published')
6
7     def __str__(self):
8         return self.question_text
9
10 class Choice(models.Model):
11     question = models.ForeignKey(Question, on_delete=models.CASCADE)
12     choice_text = models.CharField(max_length=200)
13     votes = models.IntegerField(default=0)
14
15     def __str__(self):
16         return self.choice_text
```





The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'pollProject' with a 'pollApp' subdirectory. The 'pollApp' directory contains files like 'migrations', 'urls.py', 'views.py', 'models.py', 'tests.py', 'admin.py', 'apps.py', 'db.sqlite3', and 'manage.py'. The 'urls.py' file is selected and its content is displayed in the code editor. The code defines the app name as 'polls' and lists several URL patterns for the index, detail, results, and vote views.

```
pollApp> urls.py >...
1 from django.urls import path
2 from . import views
3
4 app_name = 'polls'
5
6 urlpatterns = [
7     path('', views.index, name='index'),
8     path('<int:question_id>', views.detail, name='detail'),
9     path('<int:question_id>/results/', views.results, name='results'),
10    path('<int:question_id>/vote/', views.vote, name='vote'),
11 ]
```



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the same project structure as the first screenshot. The 'views.py' file is selected and its content is displayed in the code editor. The code defines three views: 'index', 'detail', and 'results'. The 'index' view displays a list of questions, the 'detail' view displays a single question and its choices, and the 'results' view displays the results for a specific question.

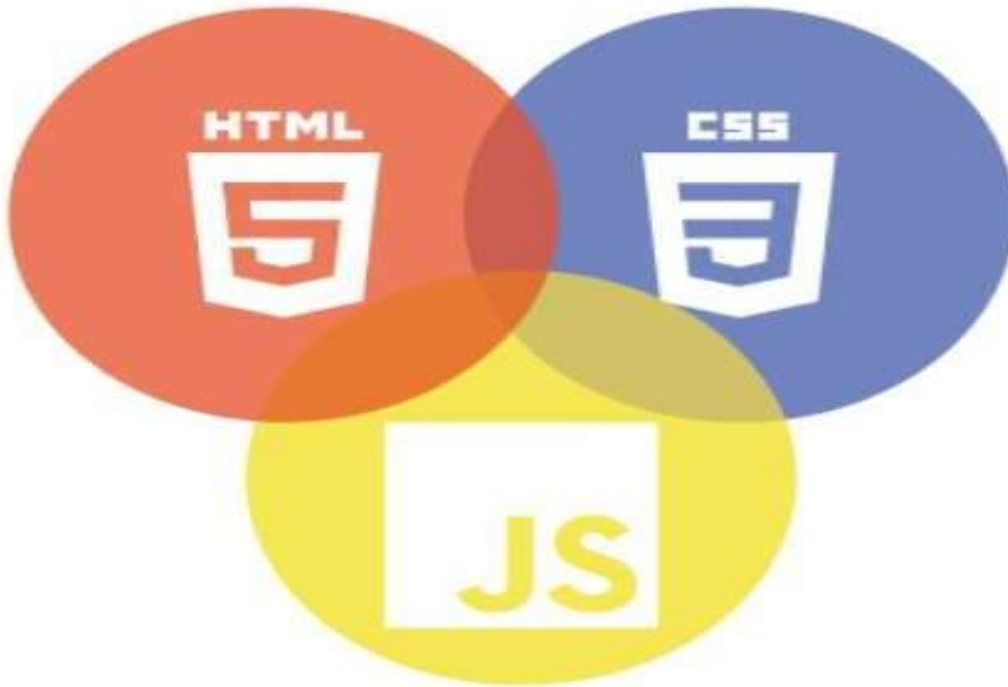
```
pollApp> views.py >...
1 from django.shortcuts import render, get_object_or_404
2 from django.http import HttpResponse, HttpResponseRedirect
3 from django.template import loader
4 from django.urls import reverse
5 from django.http import HttpResponseRedirect
6
7 from .models import Question, Choice
8
9 # Get questions and display those questions
10 def index(request):
11     latest_question_list = Question.objects.order_by('-pub_date')[:5]
12     context = {'latest_question_list': latest_question_list}
13     return render(request, 'polls/index.html', context)
14
15 # Show question and choices
16 def detail(request, question_id):
17     try:
18         question = Question.objects.get(pk = question_id)
19     except Question.DoesNotExist:
20         raise Http404('Question does not exist')
21     return render(request, 'polls/detail.html',
22                 {'question': question})
23
24 # Get question and display results
25 def results(request, question_id):
26     question = get_object_or_404(Question, pk = question_id)
27     return render(request, 'polls/results.html',
28                 {'question': question})
29
```

## Future Enhancements

Future enhancements in a voting application using the Django framework, several key features and improvements can be considered based on the information from the provided sources,

1. **Asynchronous Programming:** Implementing asynchronous programming can enhance the performance of the application by allowing tasks to run concurrently, improving responsiveness and scalability.
2. **Microservices Architecture:** Adopting a microservices architecture can make the application more modular, easier to maintain, and scalable by breaking it into smaller, independent services that communicate with each other.
3. **Serverless Computing:** Utilizing serverless computing can optimize resource utilization and reduce costs by enabling automatic scaling and only paying for actual usage, enhancing the application's efficiency and cost-effectiveness.

## ***TECHNOLOGY USED***



## Conclusion

To create a voting application using Django, one should have a solid understanding of Python programming, Django framework, HTML, CSS, and Bootstrap. The development process involves creating a new Django project, creating a Django app, defining models, creating views, defining templates, and creating URLs.

The application can be further enhanced with features such as real-time results, a user-friendly interface, and a secure database design. It can also include an admin panel for managing elections, candidates, and user accounts.

Overall, a voting application using the Django framework is a powerful and flexible solution for creating online voting systems that can cater to various use cases and requirements.