VOTING APPLICATION USING DJANGO FRAMEWORK

PROJECT TITLE

TO BUILD A E-VOTING APPLICATION USING DJANGO FRAMEWORK

VOTING APPLICATION USING DJANGO FRAMEWORK

INTRODUCTION

The system is built fully in Django Framework in back-end and HTML, CSS in front-end.

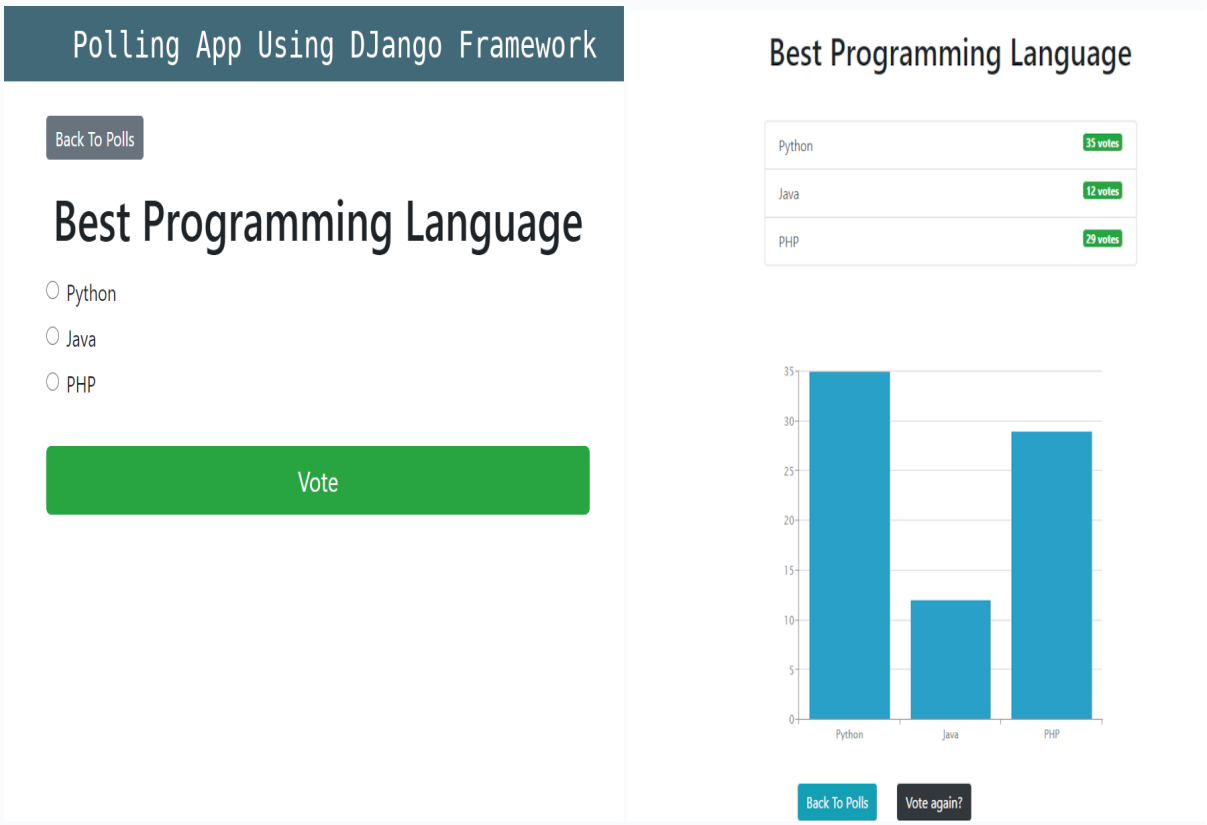
It contains all the features of any voting system such as user login, register to the system, manage user's profile and most importantly the main screen which contains voting and polls details.

It displays all the lists of polls from which users can select one and vote according to their choice.

After the voting process, users can also see the overall result of the poll which is displayed by a beautiful Interactive Charts.

There is also an admin panel from where all the polls and their questions-options can be managed.

Admin also has the control over all user's info and their voting details.



ABOUT ONLINE VOTING SYSTEM DJANGO PROJECT

In particular, this e-voting system project in Python Django focuses mainly on dealing with online voting, and voter-candidate information.

Also, the system displays selective overall data using graphical representations. In addition, the system allows managing voters' records.

Evidently, this project is divided into two categories: Voter, and Admin Panel. In an overview of this web application, a voter can simply register into the system.

Initially, the system only allows a voter to cast a vote and view his/her ballot.In fact, the system restricts users after casting a vote. This means that one user can only cast a vote once, but he/she can select up to 20 candidates. Besides, a voter can only list out his/her ballot which displays the name of voted candidates.

ADMIN PANEL

An administrator has full control over the system. He/she can manage voters, candidates, positions, and more. Here, each and every section has its own respective details such as name and other important details.

The very first step of the management of this system is to set up positions and candidates. There are minor fields for each such as name, maximum votes, and bio. Both sections play an important role in maintaining the proper flow of online voting.

As each of these records falls under different sections. Here, the position is simply referred to as the candidate's position for the upcoming election.

In order to add candidates, the admin has to provide his/her position, name, and bio with a single photo.

VOTER MANAGEMENT AND VOTES

Moving toward the voter management side, an admin has to provide various details. This includes the name of the voter, email address, phone number, and password for setting up login credentials.

As mentioned earlier, this particular thing can be done by the users themselves by registering into the system. In fact, the system does not require any other verification on both sides for registering a voter.

Additionally, each vote is counted and managed in a systematic way which an admin can view from his/her panel.

After submitting votes by the voters, an admin can simply list out all the vote records such as voter's name, candidate voter for, and position.

Here, the user can perform search queries under each and every section.

Also, an admin can download vote results in PDF format. Besides, an admin can view the ballot position of each and update the election title anytime.

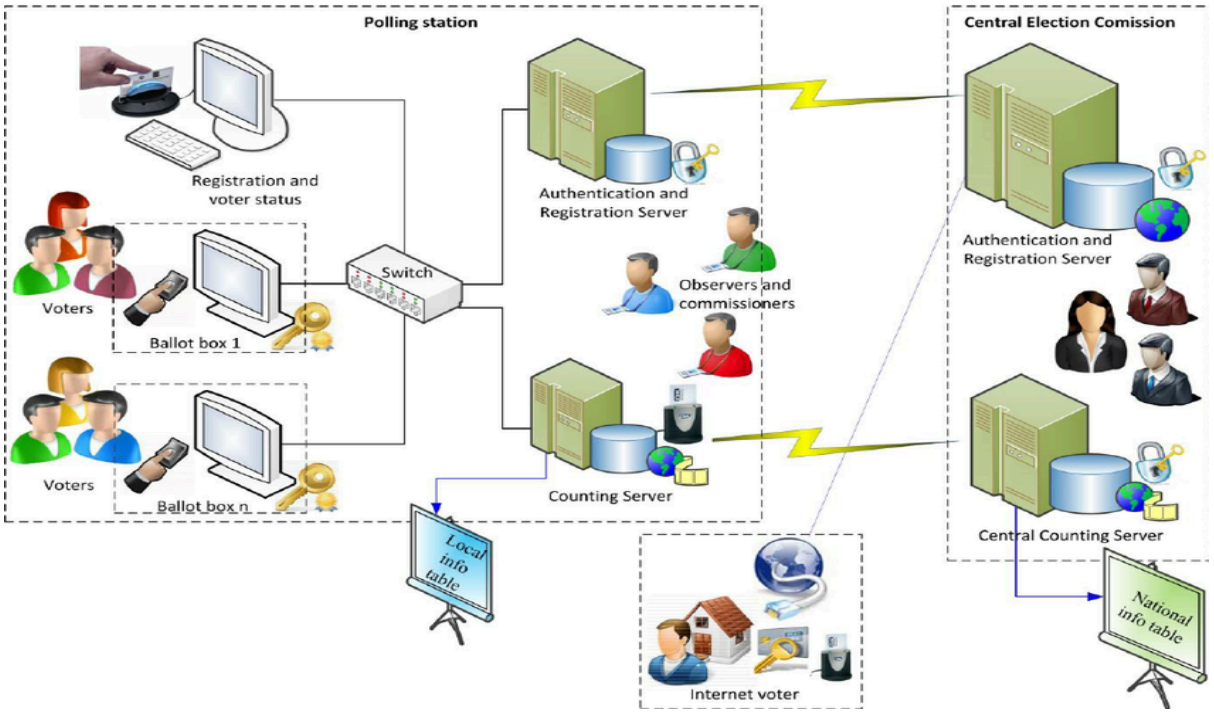
Last but not least, a clean and simple dashboard is presented with various colour combinations for greater user experience while using this Online Voting System Project in Python Django Framework.

For its UI elements, a free open-source CSS framework; Bootstrap is on board with some Vanilla CSS too. Presenting a new Online Voting System Project in Python Django which includes an admin panel with a voter's panel that contains all the essential features to follow up, and a knowledgeable resource for learning purposes.

AVAILABLE FEATURES

- Voter's Panel
- Admin Panel
- Cast Votes
- Select Multiple Candidates
- View Personal Ballot
- Voter Management
- Candidate Management
- Set up Positions

- List Vote Details
- Votes Tally
- Download Vote Results (PDF)
- View Ballot Position
- Update Election Title



HOW TO DEVELOP THE PROJECT

Creating a voting application using the Django framework can be a great project to learn about web development and Django's capabilities.

Here's a simplified guide to get you started:

1.Setup Django:

First, make sure you have Django installed. You can install it using pip:

```
pip install django
```

2.Create a Django Project:

Create a new Django project using the following command:

```
django-admin startproject voting_app
```

3.Create a Django App:

Create a new Django app within your project:

```
cd voting_app
python manage.py startapp polls
```

4.Define Models:

Define your models in the `polls/models.py` file.
For a simple voting application, you might have models for `Question` and `Choice.`

5.Create Database Tables:

Run migrations to create the necessary database tables:

```
python manage.py makemigrations
python manage.py migrate
```

6.Create Views:

- Define views in the `polls/views.py` file.
- Views handle HTTP requests and return responses.
- You'll need views for listing questions, displaying details of a question, handling voting, etc.

7.Create URLs:

Define URL patterns in the `polls/urls.py` file to map views to specific URLs.

8.Create Templates:

Create HTML templates in the `polls/templates` directory for rendering the user interface. You'll need templates for listing questions, displaying question details, and possibly for the voting form.

9.Write Business Logic:

Implement the logic for retrieving questions, choices, handling votes, etc., in your views and models.

10.Handle Forms:

If you have forms (e.g., for submitting votes), handle form processing in your views.

11.Static Files:

If you have static files (e.g., CSS, JavaScript), place them in the `polls/static` directory.

12.Templates:

Ensure your templates are correctly referencing static files and are rendering the necessary data from views.

13.Testing:

Write tests for your application to ensure it behaves as expected. Django provides a robust testing framework.

14.Run the Development Server:

Start the Django development server to see your application in action:

```
python manage.py runserver
```

15.Access the Application:

Visit `http://127.0.0.1:8000/` in your web browser to access your voting application.

16.Deploy:

Once your application is ready, deploy it to a web server to make it accessible to others.

- ❖ Remember that this is just a high-level overview. Each step involves more detailed implementation and might require further research based on your specific requirements.

- ❖ Django's official documentation is an excellent resource for detailed information on each aspect of building a Django application.

KEY FEATURES

✅ User-friendly Interface:

- ➢ Create a user-friendly and intuitive interface for voters to easily select their choices.

✅ **Secure Database:**

➢Safeguard voting data with a secure database design and protect against fraudulent activities.

✅ **Real-time Results:**

➢Display real-time voting results to keep participants and stakeholders informed.

✅ **Admin Panel:**

➢Build an admin panel to manage elections, candidates, and user accounts efficiently.

PROGRAM

Creating a full-fledged online voting application involves various components, including frontend development (HTML/CSS/JavaScript) for the user interface, backend development (Python) for server-side logic, and database integration.

Backend (Python with Django):

1.Setup Django:

Follow the steps mentioned earlier to set up a Django project and app.

2.Define Models:

Define models to represent the data structure of your voting application. For example:

```
# models.py

from django.db import models

class Question(models.Model):

    question_text = models.CharField(max_length=200)

    pub_date = models.DateTimeField('date published')

 class Choice(models.Model):

question=models.ForeignKey(Question,
on_delete=models.CASCADE)

    choice_text = models.CharField(max_length=200)

    votes = models.IntegerField(default=0)
```

3.Views and URLs:

Create views to handle HTTP requests and corresponding URLs to map those views.

4.Forms:

Create forms for user interaction (e.g., voting).

5.Business Logic:

Implement logic for retrieving questions, choices, handling votes, etc.

Frontend (HTML/CSS/JavaScript):

1.HTML Templates:

Create HTML templates for rendering the user interface. You'll need templates for listing questions, displaying question details, and voting forms.

```
<!-- voting.html -->

<!DOCTYPE html>

<html>

<head>

   <title>Online Voting Application</title>

     <link rel="stylesheet" type="text/css" href="{% static 'styles.css' %}">

</head>

<body>
```

```
    <h1>Online Voting Application</h1>
    <!-- Display questions here -->
</body>
</html>
```

2.CSS Styling:

Style your HTML templates using CSS for better presentation.

```
/* styles.css */
body {
    font-family: Arial, sans-serif;
    background-color: #f2f2f2;
}
h1 {
    color: #333;
    text-align: center;
}
```

3.JavaScript:

Implement any client-side logic using JavaScript if needed.

Database:

Integrate your Django application with a database (e.g., SQLite, PostgreSQL) to store questions, choices, and votes.

Deployment:

Deploy your application to a web server (e.g., Heroku, PythonAnywhere) to make it accessible online.

Security Considerations:

Ensure proper authentication and authorization mechanisms to prevent unauthorised access to voting features.

Testing:

Write tests for your application to ensure it behaves as expected, both frontend and backend components.

VOTING APPLICATION SOURCE CODE

pollApp - 📂migrations

📄0001_initial.py

```
from django.db import migrations, models
import django.db.models.deletion

class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [ migrations.CreateModel( name='Question',
   fields=[('id', models.BigAutoField(auto_created=True, primary_key=True,
serialize=False,verbose_name='ID')),('question_text',models.CharField(m
ax_length=('pub_date',models.DateTimeField(verbose_name='date
published')),]
        migrations.CreateModel(
            name='Choice',
            fields=[
                                        ('id',   models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('choice_text', models.CharField(max_length=200)),
                ('votes', models.IntegerField(default=0)),
                                                                        ('question',
models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
to='pollApp.question')),
            ],
        ),
    ]
```

📄admin.py

```
from django.contrib import admin
from .models import Question, Choice

admin.site.site_header = "The Poll Mall"
admin.site.site_title = "Voting Admin Area"
```

```
admin.site.index_title = "Welcome to our Voting 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)
```

📄app.py

```
from django.apps import AppConfig

class PollappConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'pollApp'
```

📄models.py

```
from django.db import models
# Question class

class Question(models.Model):
    question_text = models.CharField(max_length = 300)
    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
```

📄tests.py

```
from django.test import TestCase

# Create your tests here.
```

📄urls.py

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

app_name = 'polls'

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

📄views.py

```
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from django.urls import reverse
from django.http import Http404

from .models import Question, Choice

# Get questions and display those questions
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 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 qerstion choice
def vote(request, question_id):
```

```
    question = get_object_or_404(Question, pk = question_id)
    try:
                        selected_choice = question.choice_set.get(pk =
request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        return render(request, 'polls/detail.html',
                    { 'question': question,
                     'error_message': 'You did not select a choice.'})
    else:
        selected_choice.votes += 1
        selected_choice.save()
            return HttpResponseRedirect(reverse('polls:results', args =
(question.id,)))
```

📁pollProject

📄 asgi.py

```
"""
ASGI config for pollProject project.

It exposes the ASGI callable as a module-level variable named
``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/
"""
import os
from django.core.asgi import get_asgi_application
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'pollProject.settings')
application = get_asgi_application()
```

📄settings.py

```
"""
Django settings for pollProject project.
Generated by 'django-admin startproject' using Django 4.2.4.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""
import os
from pathlib import Path

# Build paths inside the project like this: `BASE_DIR` / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
#                                                                 See
https://docs.djangoproject.com/en/4.2/howto/deployment/checkli
st/
# SECURITY WARNING: keep the secret key used in production
secret!
SECRET_KEY = YOUR OWN KEY# SECURITY WARNING: don't run
with debug turned on in production!
DEBUG = True
ALLOWED_HOSTS = []
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
```

```
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'pollApp',
    'landingPage'
]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
ROOT_URLCONF = 'pollProject.urls'
TEMPLATES = [
    {
                                                        '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',
            ],
        },
    },
]
```

```
WSGI_APPLICATION = 'pollProject.wsgi.application'
# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
```

```
]
# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/
STATIC_URL = 'static/'
# Default primary key field type
#
https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-f
ield
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

📄urls.py

```
"""
URL configuration for pollProject project.

The `urlpatterns` list routes URLs to views. For more information
please see:
    https://docs.djangoproject.com/en/4.2/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, include

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

📄index.html

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

<div class="card text-center">
    <div class="card-body" >
      <h1> Welcome to Poll Mall!</h1>
    <p>This is my first Django
        Project after long time!</p>
    <a class="btn btn-dark" href = "{% url 'polls:index' %}">
       View Available Polls! </a>
    </div>
</div>{% endblock %}
```

📄_navbar.html

```
<nav class="navbar navbar-dark bg-primary mb-">
    <div class="container" >
    <a class="navbar-brand" href="/">The Poll Mall</a>
```

```
    </div>
</nav>
```

📄detail.html

```
{% 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 %}
```

📄index.html

```
{% 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 %}
```

📄results.html

```
{% 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 %}
```

📄base.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>Poll Mall {% block title %}{% endblock %}</title>
</head>

<body>
        <!--NavBar-->
```

```
        {% include 'partials/_navbar.html'%}
        <div class ="container">
                <div class ="row">
                        <div class =".col-md-3 m-auto">
                                {% block content %}{% endblock %}
                        </div>
                </div>
        </div>
</body>
</html>
```

📄manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys

def main():
    """Run administrative tasks."""
                    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'pollProject.settings')
    try:
                                from    django.core.management    import
execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
              "available on your PYTHONPATH environment variable? Did
you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
```

```
if __name__ == '__main__':
    main()
```

📄Pipfile

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
django = "*"

[dev-packages]

[requires]
python_version = "3.10"
```

📄Pipfile.lock

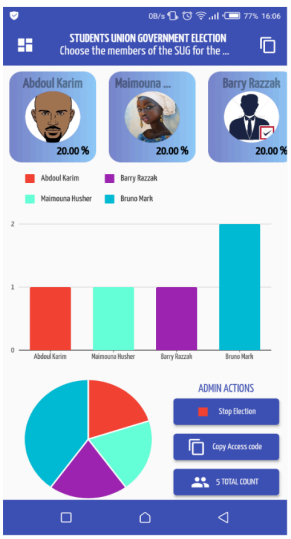
```
{
    "_meta": {
        "hash": {
                                                                "sha256":
"7e6dca07b964c2888324e576ba6c1bc240d74a27b75619fc88bca2
ee3979baf8"
        },
        "pipfile-spec": 6,
        "requires": {
            "python_version": "3.10"
        },
        "sources": [
            {
```

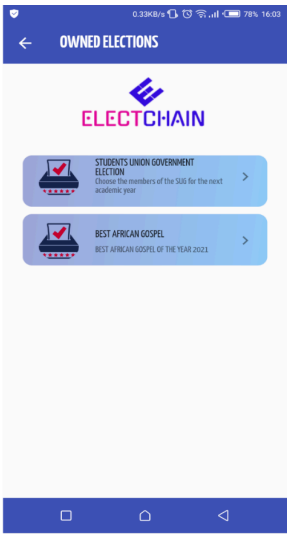
```
            "name": "pypi",
            "url": "https://pypi.org/simple",
            "verify_ssl": true
        }
    ]
},
"default": {
    "asgiref": {
        "hashes": [

"sha256:89b2ef2247e3b562a16eef663bc0e2e703ec6468e2fa8a5cd61cd449786d4f6e",

"sha256:9e0ce3aa93a819ba5b45120216b23878cf6e8525eb3848653452b4192b92afed"
        ],
        "markers": "python_version >= '3.7'",
        "version": "==3.7.2"
    },
    "django": {
        "hashes": [

"sha256:7e4225ec065e0f354ccf7349a22d209de09cc1c074832be9eb84c51c1799c432",

"sha256:860ae6a138a238fc4f22c99b52f3ead982bb4b1aad8c0122bcd8c8a3a02e409d"
        ],
        "index": "pypi",
        "version": "==4.2.4"
    },
    "sqlparse": {
        "hashes": [
```

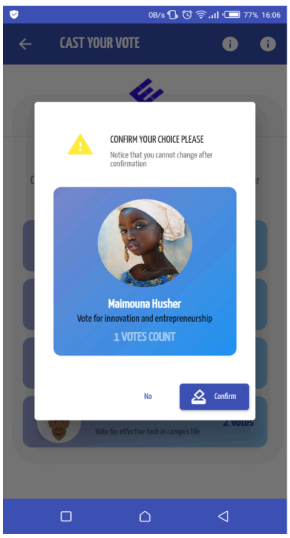
```
"sha256:5430a4fe2ac7d0f93e66f1efc6e1338a41884b7ddf2a350ce
dd20ccc4d9d28f3",

"sha256:d446183e84b8349fa3061f0fe7f06ca94ba65b426946ffebe
6e3e8295332420c"
            ],
            "markers": "python_version >= '3.5'",
            "version": "==0.4.4"
        },
        "typing-extensions": {
            "hashes": [

"sha256:440d5dd3af93b060174bf433bccd69b0babc3b15b1a8dca
43789fd7f61514b36",

"sha256:b75ddc264f0ba5615db7ba217daeb99701ad295353c45f9
e95963337ceeeffb2"
            ],
            "markers": "python_version < '3.11'",
            "version": "==4.7.1"
        },
        "tzdata": {
            "hashes": [

"sha256:11ef1e08e54acb0d4f95bdb1be05da659673de4acbd21bf9
c69e94cc5e907a3a",

"sha256:7e65763eef3120314099b6939b5546db7adce1e7d6f2e17
9e3df563c70511eda"
            ],
            "markers": "sys_platform == 'win32'",
            "version": "==2023.3"
```

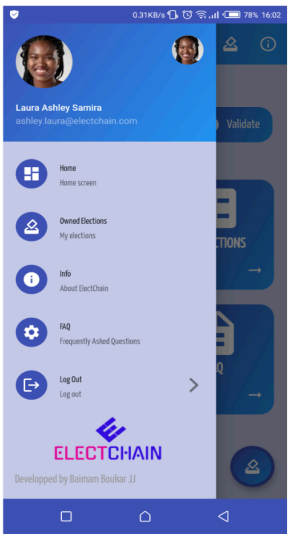
```
        }
    },
    "develop": {}
}
```

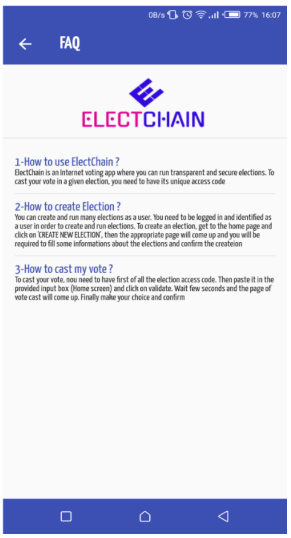
OUTPUT











ADVANTAGES OF VOTING APPLICATIONS

1.Convenience:

Online voting applications provide convenience to voters, allowing them to cast their votes from anywhere with an internet connection, eliminating the need to travel to polling stations.

2.Accessibility:

Voting applications can be designed to be accessible to people with disabilities, providing features such as screen readers and alternative input methods, thus ensuring inclusivity in the voting process.

3.Increased Participation:

By removing barriers such as geographical distance and time constraints, online voting applications can potentially increase voter participation, leading to more representative and democratic outcomes.

4.Cost-Efficiency:

Online voting can be more cost-effective than traditional voting methods, as it reduces the need for physical infrastructure, paper ballots, and manual labour associated with traditional voting processes.

5.Faster Results:

Online voting applications can streamline the voting process, leading to faster tabulation and announcement of results compared to manual counting of paper ballots.

6.Accuracy and Integrity:

Online voting systems can incorporate built-in security measures to ensure the accuracy and integrity of the voting process, including encryption, authentication mechanisms, and audit trails to detect and prevent fraud.

7.Flexibility and Customization:

Voting applications can be customised to accommodate various voting methods (e.g., plurality voting, ranked-choice voting), languages,

and voting periods, providing flexibility to meet the specific needs of different elections or organisations.

8.Environmental Sustainability:

By reducing the use of paper and physical resources, online voting applications contribute to environmental sustainability by minimising waste and carbon footprint associated with traditional voting methods.

9.Transparency:

Online voting systems can offer transparency by providing access to voting records, audit logs, and verifiable encryption methods, allowing voters and stakeholders to verify the integrity of the voting process.

10.Scalability:

Online voting applications can scale to accommodate a large number of voters and simultaneous voting events, making them suitable for elections of various scales, from community elections to national or international elections.

DISADVANTAGES OF VOTING APPLICATION

1.Security Concerns:

One of the primary disadvantages of voting applications is the potential for security breaches and manipulation. Hackers could exploit vulnerabilities in the application to tamper with election results or compromise voter privacy, undermining the integrity and trustworthiness of the entire voting process.

2.Digital Divide:

Not all voters may have access to the necessary technology or internet connectivity required to use voting applications. This can exacerbate existing inequalities and disenfranchise certain groups of voters, leading to concerns about the fairness and inclusivity of the electoral process.

3.Vulnerability to Cyber Attacks:

Voting applications are susceptible to various cyberattacks, including distributed denial-of-service (DDoS) attacks, malware injections, and phishing attempts. These attacks can disrupt the voting process, compromise voter data, and erode public confidence in the security of online elections.

4.Complexity and Technical Challenges:

Developing and maintaining secure and reliable voting applications requires sophisticated technical expertise and infrastructure. Ensuring compatibility across different devices and platforms, addressing software bugs and glitches, and maintaining compliance with evolving security standards can pose significant challenges for election administrators and software developers.

5.Trust and Transparency:

Despite efforts to enhance transparency and accountability in online voting systems, some voters may remain sceptical about the accuracy, fairness, and transparency of the electronic voting process. Concerns about the verifiability of votes, the anonymity of voters, and the potential for manipulation may deter participation and erode trust in democratic institutions.

THANK YOU!