An online voting system, also known as e-voting, is a digital platform that allows eligible voters to cast their votes remotely using the internet. It is a modern alternative to traditional paper-based voting methods. With an online voting system, voters can access the voting process through a secure website or a dedicated voting application.

About Python Django Online Voting System

The objective of a Python Django online voting system is to provide a convenient and efficient method for eligible voters to cast their votes remotely using the internet.

Prerequisite For Online Voting System Using Python Django

A solid understanding of the Python programming language and the Django web framework is necessary.

A strong understanding of HTML, CSS, and JavaScript is required to develop the project’s user interface.

Relational Database: You will need to have a good understanding of relational databases, such as SQLite, MySQL, or PostgreSQL, to create and manage the database for the Online voting system project.

Download Python Django Online Voting System Project

Please download the source code of the Python Django Online Voting System: Python Django Online Voting System Project Code.

Project Setup

Minimum system configuration:

The operating system requirements include Windows 7 or later, macOS 10.11 or later, or a modern operating system.

Linux distribution.

Processor: Intel Core i3 or equivalent.

RAM: 4 GB or more

Disk Space: 5 GB or more.

Browsers such as Google Chrome, Mozilla Firefox, or Microsoft Edge can be used.

Visual Studio Code can be downloaded from the official website.

On the download page, you can select the suitable installer for your operating system (Windows, macOS, or Linux). After downloading the installer, run it and proceed with the installation instructions to install VS Code on your computer.

Here’s a brief explanation of each step, along with the commands to execute:

1. Python should be installed: Download and install the latest version of Python from the official website, following the installation instructions for your operating system.

2. Install pip: Download the get-pip.py script and run python get-pip.py to install pip.

3. Create a virtual environment: Run python -m venv myenv to create a new virtual environment named ‘myenv’.

4. Activate the virtual environment: Run source myenv/bin/activate on Linux/Mac or myenv\Scripts\activate on Windows to activate the virtual environment.

5. Install Django: Run pip install django to install the latest stable version of Django.

6. Verify installation: Run python -m django –version to verify that Django is installed correctly.

7. Create a new Django project: Run django-admin startproject project to create a new Django project named ‘project’.

8. Start the development server: Run python manage.py runserver to start the development server.

That’s it! A working installation of Django should now be in place, and you should be ready to start building your web application.

From django.db import models

From django.contrib.auth.models import User

From django.core.validators import MinValueValidator

# Create your models here.

Class Questions(models.Model):

User = models.OneToOneField(User, on\_delete=models.CASCADE)

Ques = models.CharField(max\_length=150)

Option1 = models.CharField(max\_length=150)

Option2 = models.CharField(max\_length=150)

Option3 = models.CharField(max\_length=150)

Option4 = models.CharField(max\_length=150)

Vote1 = models.IntegerField(default=0)

Vote2 = models.IntegerField(default=0)

Vote3 = models.IntegerField(default=0)

Vote4 = models.IntegerField(default=0)

Vote = models.IntegerField(default=False, verbose\_name=”How many object created for this questions?”)

Is\_closed = models.BooleanField(default=False)

@property

Def total\_votes(self):

Return self.vote1 + self.vote2 + self.vote3 + self.vote4

@property

Def get\_winner\_option(self):

Options = [self.vote1, self.vote2, self.vote3, self.vote4]

Max\_votes = max(options)

Winner\_index = options.index(max\_votes)

If options.count(max\_votes) > 1:

Return “It’s a tie”

Else:

If winner\_index == 0:

Return self.option1

Elif winner\_index == 1:

Return self.option2

Elif winner\_index == 2:

Return self.option3

Elif winner\_index == 3:

Return self.option4

Return

Def \_\_str\_\_(self) -> str:

Return self.ques

Class Voted(models.Model):

User = models.ForeignKey(User, on\_delete=models.CASCADE)

Voted\_question = models.ForeignKey(Questions, on\_delete=models.CASCADE)

Class UserProfile(models.Model):

User = models.OneToOneField(User, on\_delete=models.CASCADE)

Age = models.PositiveIntegerField(validators=[MinValueValidator(0)])

To create the above field in a database, run the following commands as follows:

Py manage.py makemigrations

Py manage.py migrate

7. Create a Registration system

<h2>{{ item.ques }}</h2>

<label><input type=”radio” name=”selected\_option” value=”1”>{{ item.option1 }}</label><br>

<label><input type=”radio” name=”selected\_option” value=”2”>{{ item.option2 }}</label><br>

<label><input type=”radio” name=”selected\_option” value=”3”>{{ item.option3 }}</label><br>

<label><input type=”radio” name=”selected\_option” value=”4”>{{ item.option4 }}</label><br>

<h4>Total Votes: {{item.total\_votes}}</h4>

{% if item.is\_closed %}

<p><strong>Winner: {{ item.get\_winner\_option }}</strong></p>

<button class=”vote-button” type=”button” disabled>

Voting Closed

</button>

{% else %}

<button class=”vote-button” type=”submit”>

{% if user\_profile.age > 18 %}

Vote Now

{% else %}

Not allowed

{% endif %}

</button>

{% endif %}

</form>

<br>

</div>

{% endfor %}

Views.py

@login\_required(login\_url=”login”)

Def Voting(request, pk):

User = request.user

Ques = get\_object\_or\_404(Questions, pk=pk)

# Check if the user is below 18 years old

User\_profile = UserProfile.objects.get(user=user)

If user\_profile.age < 18:

Messages.warning(request, “Voters below 18 years of age are not allowed to vote.”)

Return redirect(“votingpage”)

# Check if the user has already voted for this question

If Voted.objects.filter(user=user, voted\_question=ques).exists():

Messages.warning(request, “You have already voted for this question.”)

Return redirect(“already”)

Else:

Selected\_option = request.POST.get(‘selected\_option’)

If selected\_option in [‘1’, ‘2’, ‘3’, ‘4’]:

Setattr(ques, f’vote{selected\_option}’, getattr(ques, f’vote{selected\_option}’) + 1)

Ques.save()

Voted.objects.create(user=user, voted\_question=ques)

Messages.success(request, “Your vote has been recorded.”)

Else:

Messages.warning(request, “Invalid vote selection.”)

Return redirect(‘show’)

@login\_required(login\_url=”login”)

Def show(request):

New\_ques = Questions.objects.all()

# Check if the user’s age is less than 18 and show a warning message

User\_profile = UserProfile.objects.get(user=request.user)

If user\_profile.age < 18:

Messages.warning(request, “Voter below 18 age group is not allowed.”)

Return render(request, “app/votingpage.html”, {“new\_ques”: new\_ques, “user\_profile”: user\_profile})

Urls.py

Path(“home”, views.home, name=”home”),

Path(“votingpage/<int:pk>”, views.Voting, name=”votingpage”),

Path(“showques”, views.show, name=”show”),

10. For logout process

Views.py

@login\_required(login\_url=”login”)

Def signout(request):

Logout(request)

Return redirect(“index”)

Urls.py

Path(“logout”, views.signout, name=”logout”)

Explanation of the above snippets:

Certainly! Here’s an explanation of each view in the provided code:

1. Index(request):

This view renders the login page (login.html).

1. Register(request):

This view renders the registration page (registration.html).

1. Registration(request):

This view handles the registration process when the registration form is submitted.

It first checks if the passwords provided by the user match.

If the passwords match, a new User object is created with the provided username and email.

The password Is set and encrypted using the set\_password method.

The new user is saved to the database, and the user is redirected to the login page (index).

1. Loginview(request):

This view handles the login process when the login form is submitted.

It retrieves the username and password entered by the user.

The authenticate function is used to verify the credentials. If the authentication is successful, the user is logged in using the login function, and they are redirected to the “show” page.

If the authentication fails, an “invalid credentials” message is displayed.

1. Home(request):

This view renders the main page (votingpage.html) after the user has successfully logged in.

The @login\_required decorator ensures that only authenticated users can access this page. If a user is not logged in, they are redirected to the login page.

1. Voting(request, pk):

This view handles the process of voting for a specific question.

It first retrieves the authenticated user and the question (specified by its primary key pk) using get\_object\_or\_404.

It checks if the user has already voted for the question by searching for an existing record in the Voted model.

If the user has not voted for the question, the vote count for the question is incremented, and the vote is saved.

A new record is created in the Voted model to indicate that the user has voted for that question.

The user is then redirected to the “show” page.

1. Show(request):

This view retrieves all the questions from the Questions model and renders the “votingpage.html” template.

The retrieved questions are passed to the template as the context variable new\_ques.

1. Signout(request):

This view handles the logout process.

The logout function is called to log out the user.

The user is then redirected to the login page (index).

These views, along with the corresponding URL patterns defined in the urlpatterns list, form the routing and logic for the online voting system.

.container {

Background-color: #ffffff;

Border-radius: 5px;

Box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

Padding: 20px;

Width: 300px;

}

H1 {

Text-align: center;

}

Form {

Display: flex;

Flex-direction: column;

}

Input[type=”text”],

Input[type=”password”] {

Padding: 10px;

Margin-bottom: 10px;

Border: 1px solid #ccc;

Border-radius: 4px;

}

Button {

Padding: 10px;

Background-color: #4caf50;

Color: #ffffff;

Border: none;

Border-radius: 4px;

Cursor: pointer;

}

Button:hover {

Background-color: #45a049;

}

</style>

</head>

<body>

<div class=”container”>

<h1><u>DataFlair Voting System</u></h1>

<div class=”form-container”>

<h1>Login</h1>

<form action=”{% url ‘login’ %}” method=”post”>

{% csrf\_token %}

<input type=”text” placeholder=”Username” name=”uname”>

<input type=”password” placeholder=”Password” name=”password”>

<button type=”submit”>Login</button>

</form>

<p>Not registered? <a href=”{% url ‘register’ %}”>click here</a></p>

</div>

</div>

</body>

</html>

# Check if the user’s age is less than 18 and show a warning message

User\_profile = UserProfile.objects.get(user=request.user)

If user\_profile.age < 18:

Messages.warning(request, “Voter below 18 age group is not allowed.”)

Return render(request, “app/votingpage.html”, {“new\_ques”: new\_ques, “user\_profile”: user\_profile})

Def signout(request):

Logout(request)

Return redirect(“index”)

Urls.py

From django.urls import path,include

From . import views

Urlpatterns = [

Path(“”, views.Index, name=”index”),

Path(“register”, views.register, name=”register”),

Path(“registration”, views.Registration, name=”registration”),

Path(“home”, views.home, name=”home”),

Path(“login”, views.Loginview, name=”login”),

Path(“votingpage/<int:pk>”, views.Voting, name=”votingpage”),

Path(“showques”, views.show, name=”show”),

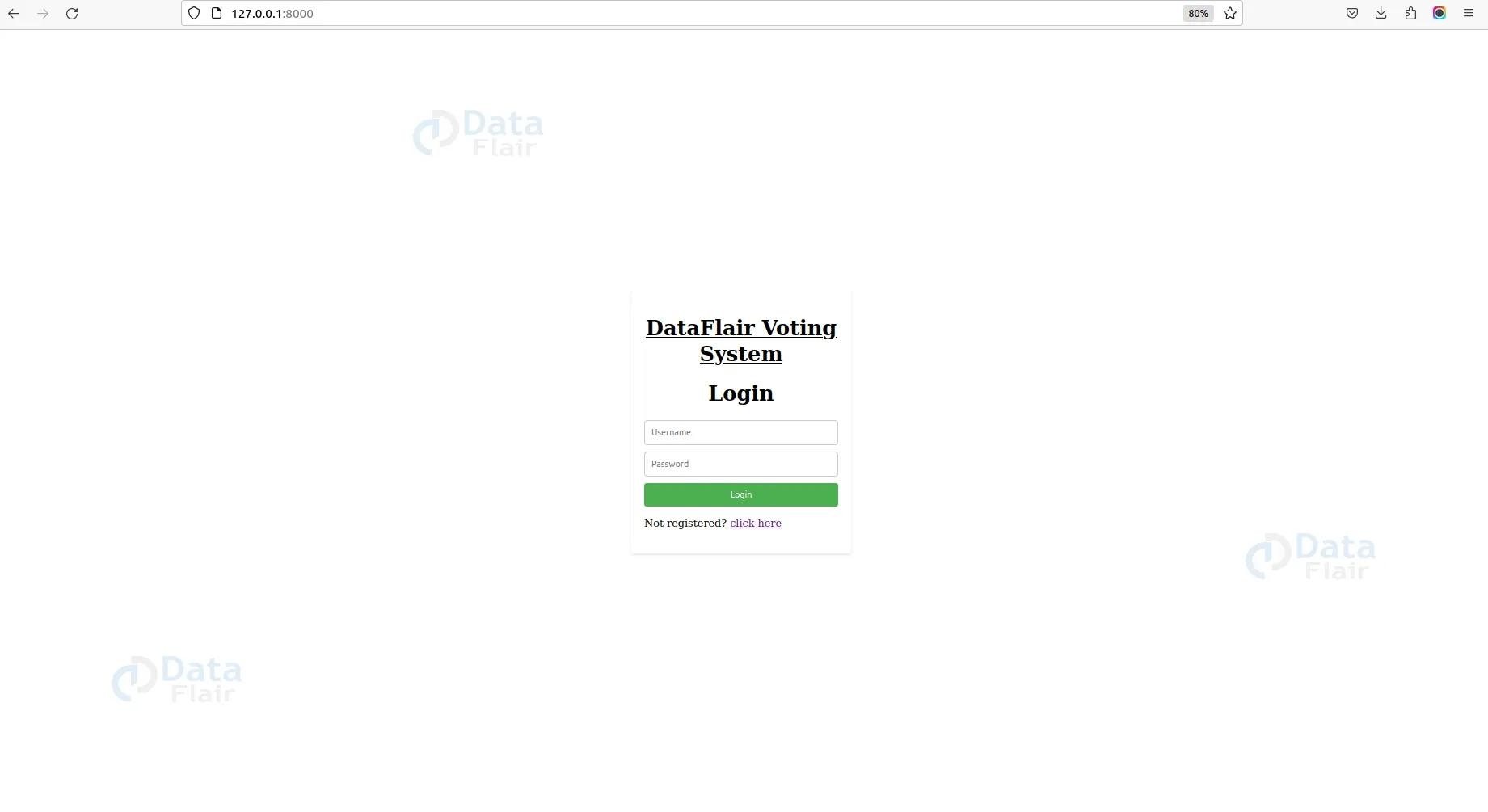
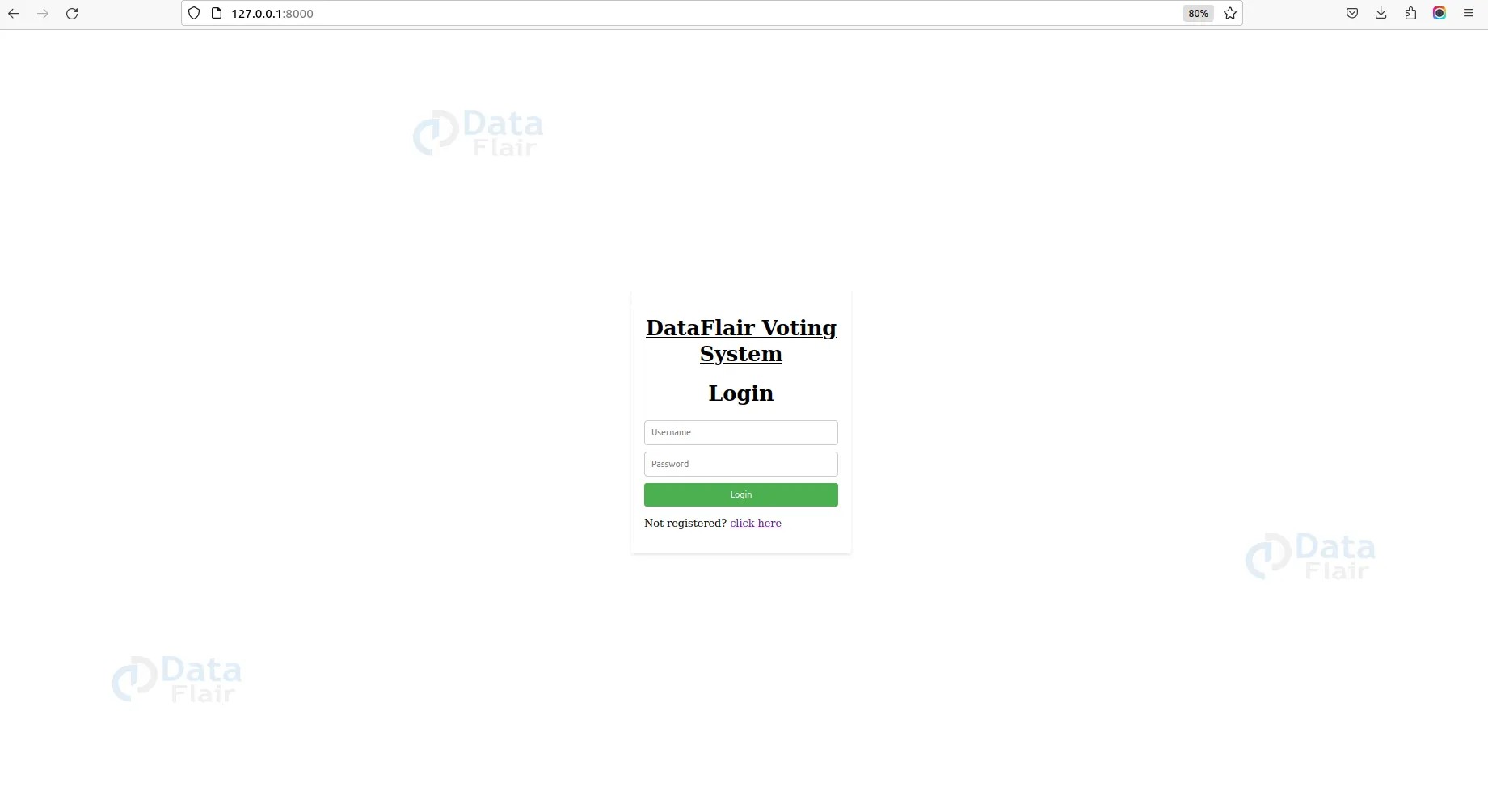
Path(“logout”, views.signout, name=”logout”),

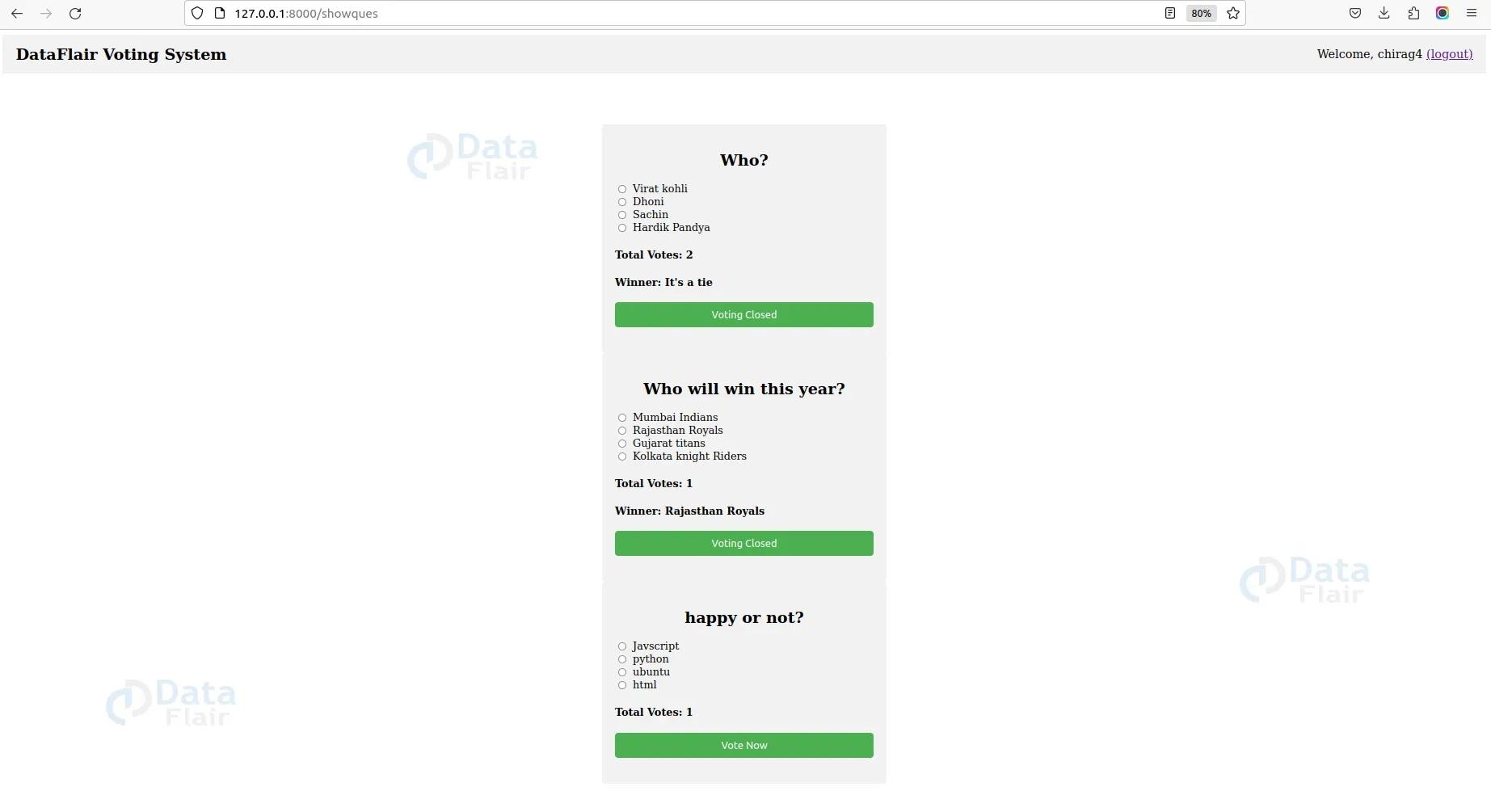
Path(“successfully”, views.successfully, name=”successfully”),

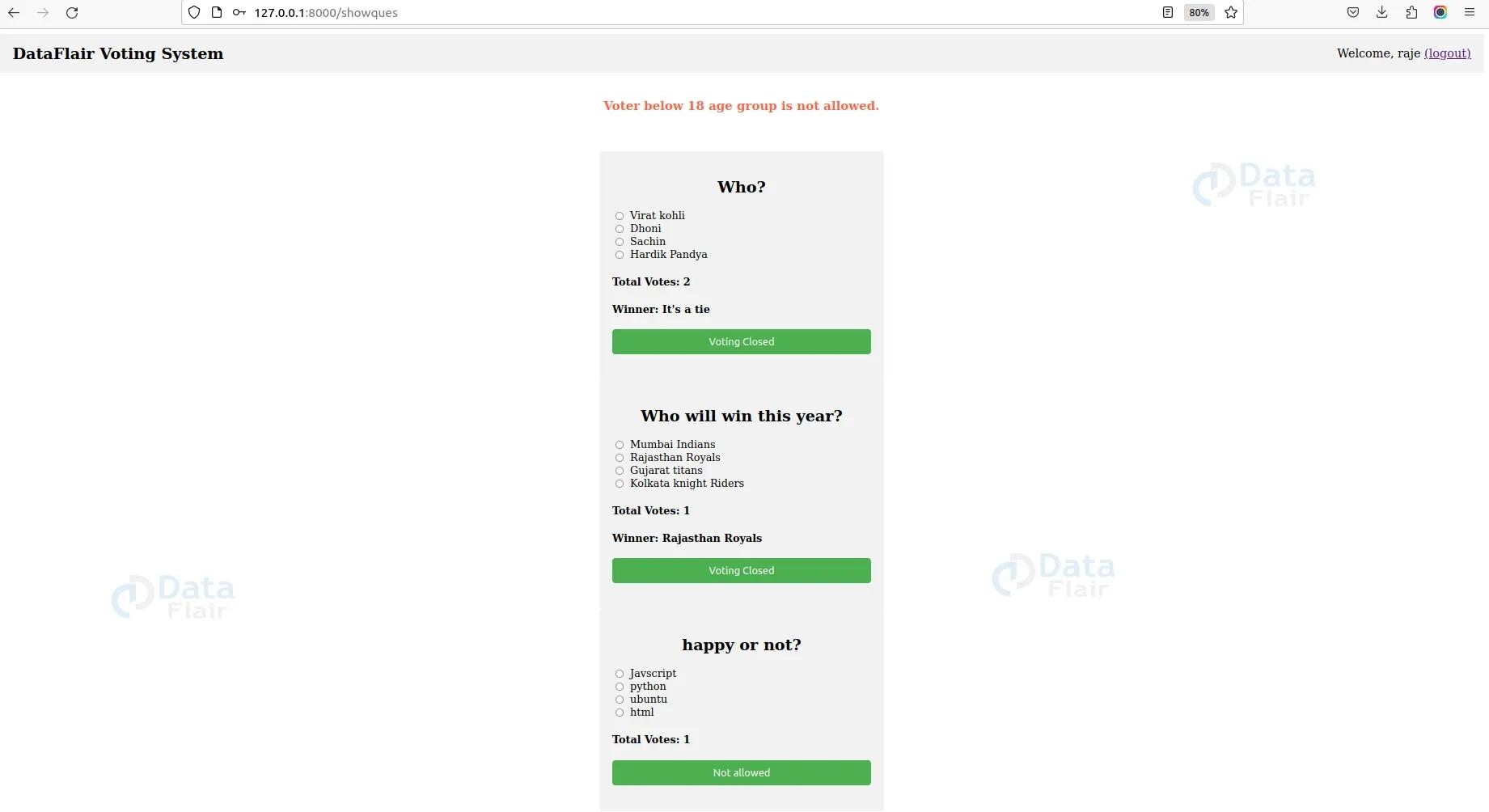
Path(“already”, views.already, name=”already”)

]

Python Django Online Voting System Output







Online voting system login output

Online voting system registration output

Online voting system project output

Voting page below age 18

Admin panel

Now how would we add data, delete, and update to the user side? Let’s come to the django inbuilt admin panel. So here are the details of the same:

1. Once you have your Django project set up and have created your models, you can use the admin panel to add data easily. Follow these steps:

2. Run python manage.py createsuperuser to create an admin user who can access the admin panel. Start the development server using python manage.py runserver. Open your browser and go to the admin panel URL (usually <http://127.0.0.1:8000/admin/>) Log in using the superuser credentials you created.

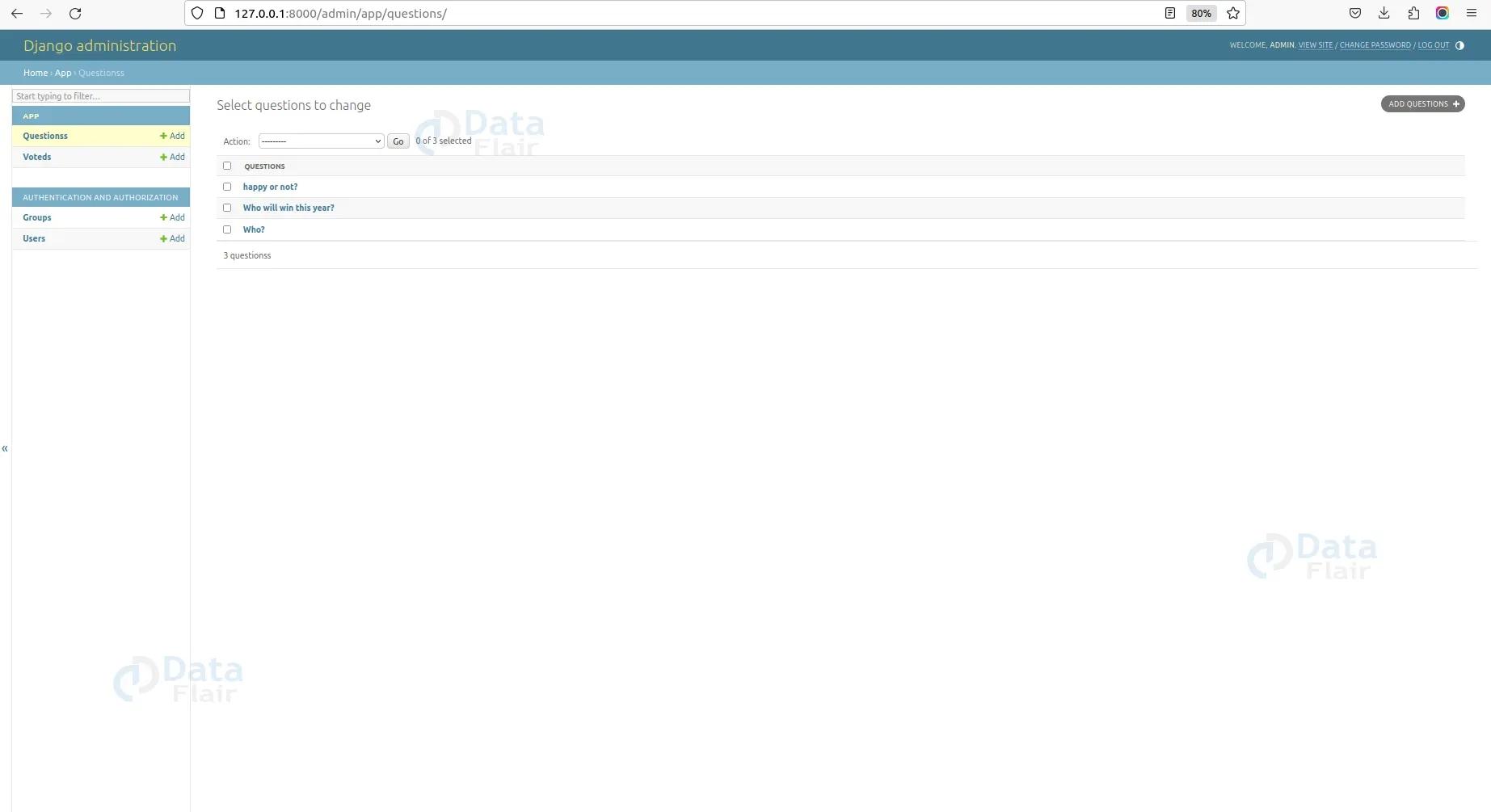
3. Then, Go to admin.py and add the following thing as follows:

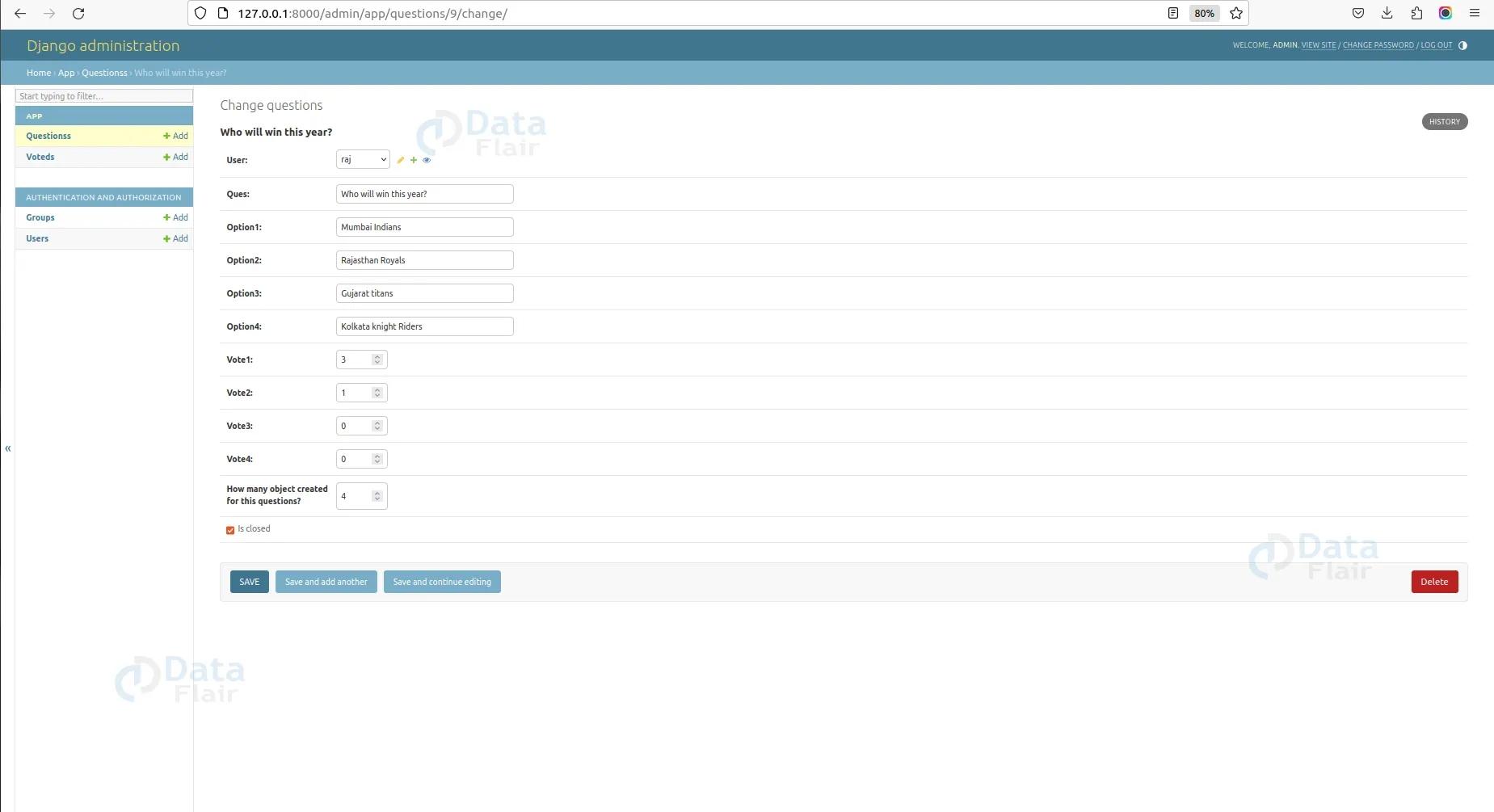
From django.contrib import admin

From .models import YourModelName

Admin.site.register(YourModelName)

Note: To display the model on the admin panel, you need to follow the third point.





Admin panel show question

Total votes admin page

Summary:

The provided code represents a basic online voting system implemented using Django framework. It includes registration, login, and voting functionalities. Users can register, log in, view and vote on questions. The system ensures authentication, prevents duplicate votes, and tracks user activity.