VYTAUTAS MAGNUS UNIVERSITY
FACULTY OF INFORMATICS
DEPARTMENT OF APPLIED INFORMATICS

INTERNET TECHNOLOGY (INF3002_EN)

Polls

Student: Dmytro Vodianytskyi IF2100075

Lab assistant: Vytautas Barzdaitis

Kaunas, 2023

Contents

1 Polls App

The Polls app is a Django app that allows users to create and vote on polls.

1.1 Directory structure

`/polls/`: is the root directory for the Polls app.

- `/migrations/`: contains database migration files.
- `/static/`: contains static files such as CSS and JavaScript files.
    - `/polls/`
        - `/img/`: contains img files of polls app
        - `/sort.js`
        - `/style_detail.css`
        - `/style_ind.css`
- `/templates/`: contains HTML templates for the Polls app.
    - `/polls/`
        - `/alrd_voted.html`
        - `/detail.html`
        - `/index.html`
        - `/results.html`
- `/admin.py`: contains code for the Polls app's admin interface.
- `/apps.py`: contains app configuration settings.
- `/models.py`: contains code for the Polls app's data models.
- `/urls.py`: contains URL routing rules for the Polls app.
- `/utils.py`: contains utility functions for the Polls app.
- `/views.py`: contains code for the Polls app's views.

1.2 /admin.py

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

admin.site.register(Question)
admin.site.register(Choice)
```

This code registers the Question and Choice models with the Django admin site.

1.3 /apps.py

```
from django.apps import AppConfig

class PollsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'polls'
```

This code provides configuration settings for the Polls app.

1.4 /models.py

```
import datetime
from django.db import models
from django.utils import timezone
from users.models import User
```

Necessary imports

```
class Question(models.Model):
question_text = models.CharField(max_length=200)
pub_date = models.DateTimeField('date published')

    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)

    def __str__(self):
        return self.question_text
```

Model to create questions, function `was_published_recently()` selects recently created questions, function `__str__` returns only 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
```

Model to create choices answers related to the question with field `question`
function `__str__` returns only choice text

```
class Vote(models.Model):
user = models.ForeignKey(User, on_delete=models.CASCADE)
question = models.ForeignKey(Question, on_delete=models.CASCADE)
choice = models.ForeignKey(Choice, on_delete=models.CASCADE)

def __str__(self):
    return f'{self.user.username} voted on {self.question.question_text}'
```

Model relates the `user`, `question` and the `choice` made by the user by fields of the same name,function `__str__` returns username of user and question text of question in which he voted

1.5 /urls.py

```
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>/remove_vote/', views.remove_vote, name='remove_vote'
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

This code defines the URL patterns for the Django application called "**polls**". The `path()` function is used to define each URL pattern.

The `''` pattern maps to the views.index function, which will display the list of all available polls. The `<int:question_id>/` pattern maps to the views.detail function, which will display the details of a specific poll.

The `<int:question_id>/results/` pattern maps to the views.results function, which will display the results of a specific poll. The `<int:question_id>/remove_vote/` pattern maps to the `views.remove_vote` function, which will allow a user to remove their vote for a specific poll. The `<int:question_id>/vote/` pattern maps to the views.vote function, which will allow a user to vote for a specific poll.

The `app_name` variable is used to set the application namespace for these URLs. This is useful in cases where multiple applications might have similar URL patterns.

1.6 /utils.py

```
def has_voted(request, poll):
    if request.session.get('has_voted_%d' % poll.id, False):
        return True
    else:
        return False
```

Function `has_voted(request, poll)` is binary function which will return `True` if the user voted in a particular poll if not `False`

1.7 /views.py

Views file with several functions that define the behavior of the application's views.

```
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponse, HttpResponseRedirect
from django.contrib.auth.decorators import login_required
from django.urls import reverse

from .models import Question, Choice, Vote
from .utils import has_voted
```

Necessary imports

```
@login_required
def index(request):
    order_by = request.GET.get('order_by', '-pub_date')
    latest_question_list = Question.objects.order_by(order_by)[:]
    context = {
        'latest_question_list': latest_question_list,
    }
    return render(request, 'polls/index.html', context)
```

`index(request)`: This function retrieves a list of the latest questions ordered by the '`pub_date`' field and renders it in the '`index.html`' template.

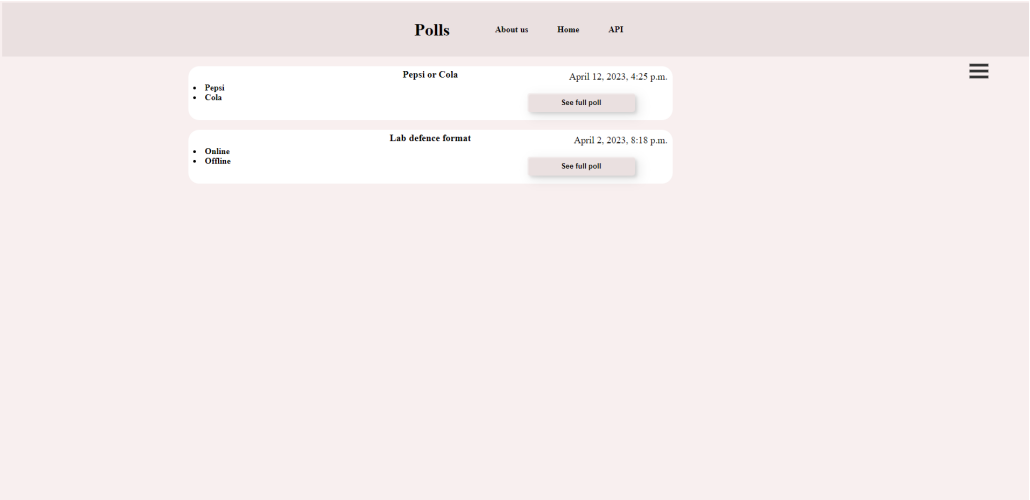

Figure 1: Screenshot of the index page

```
@login_required
def detail(request, question_id):
    question = get_object_or_404(Question, pk = question_id)
    conext = {
        'question': question
    }
    return render(request, 'polls/detail.html', conext)
```

`detail(request, question_id)`: This function retrieves the details of a specific question identified by the given `question_id` and renders it in the '`detail.html`' template.

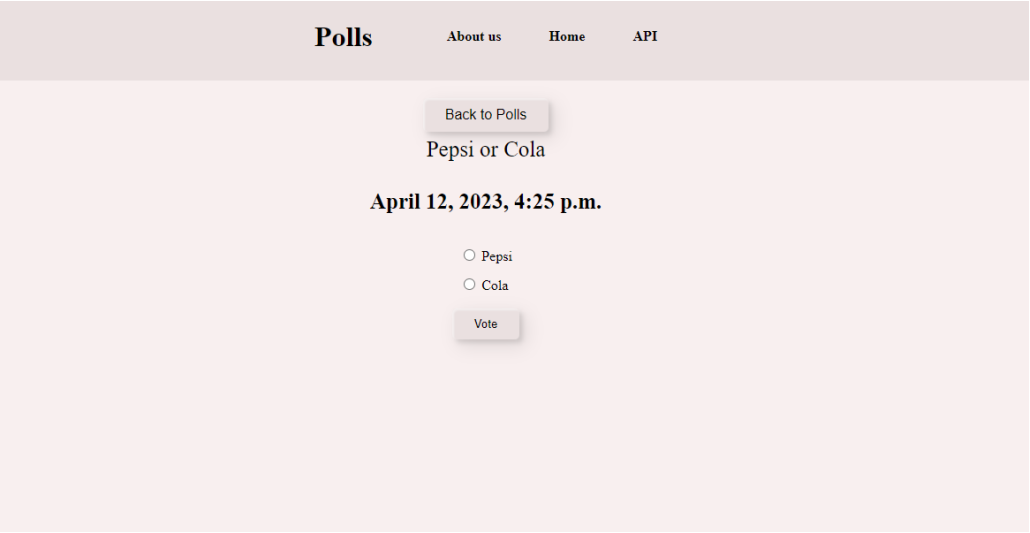

Figure 2: Screenshot of the detail page

```
@login_required
def results(request, question_id):
    question = get_object_or_404(Question, pk = question_id)
    context = {
        'question' : question,
    }
    return render(request, 'polls/results.html', context)
```

`results(request, question_id)`: This function retrieves the results of a specific question identified by the given `question_id` and renders it in the '`results.html`' template.

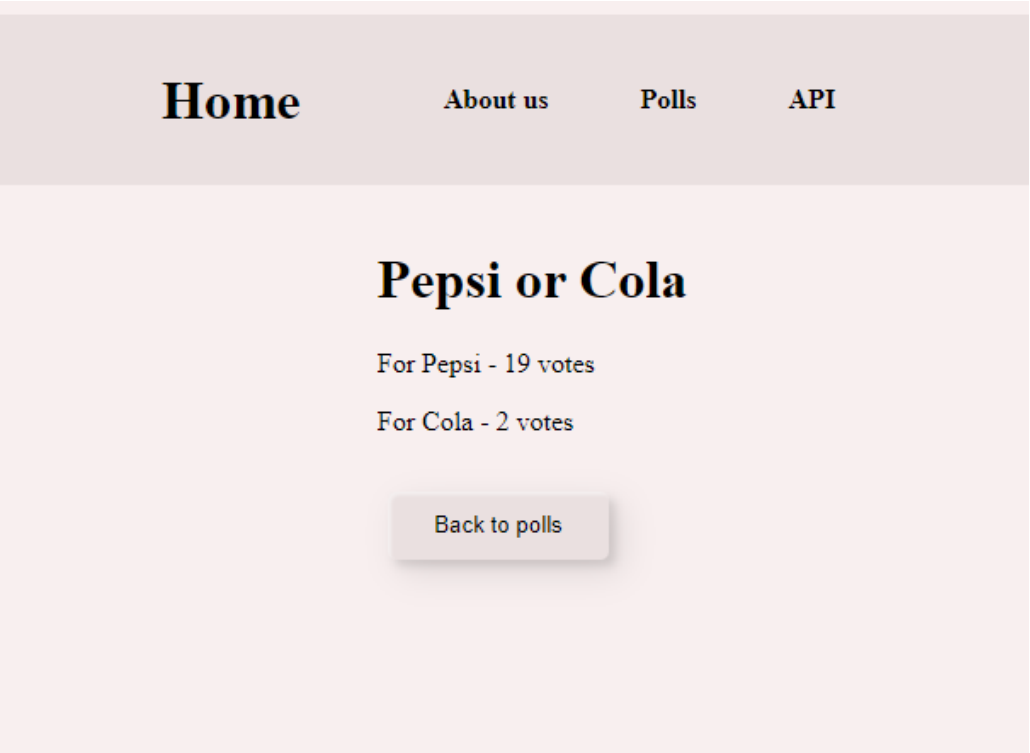

Figure 3: Screenshot of the results page

```
@login_required
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    context = {
        'question': question,
        'error_messege': 'You didnt select a choice.'
    }

    # If user doesnt vote in poll
    if Vote.objects.filter(user=request.user, question=question).exists():
        return render(request, 'polls/alrd_voted.html',
        {'question': question})

    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice']
    except (KeyError, Choice.DoesNotExist):
        return render(request, 'polls/detail.html', context)
    else:
        selected_choice.votes += 1
        selected_choice.save()


        vote = Vote(user=request.user, question=question,
                    choice=selected_choice)

        vote.save()

        return HttpResponseRedirect(reverse('polls:results',
                args=(question.id,)))
```

`vote(request, question_id)`: This function retrieves a specific question identified by the given `question_id`, checks if the user has already voted in this poll, saves the user's vote, and updates the choice count. If the user has already voted, it redirects to '`alrd_voted.html`' template.

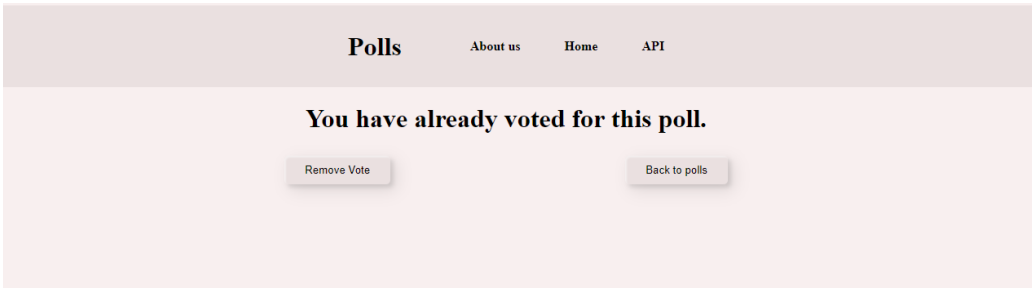

Figure 4: Screenshot of the results page

```
@login_required
def remove_vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)

    # Find the vote made by the user for the current question
    try:
        vote = Vote.objects.get(user=request.user, question=question)
    except Vote.DoesNotExist:
        return HttpResponse('You have not voted in this poll.')

    # Remove the vote and update the choice count
    choice = vote.choice
    choice.votes -= 1
    choice.save()
    vote.delete()

    return HttpResponseRedirect(reverse('polls:detail',
            args=(question.id,)))
```

`remove_vote(request, question_id)`: This function removes the user's vote for a specific question identified by the given `question_id`, updates the choice count, and redirects to '`detail.html`' template.

All functions use the Django's `render()` and `HttpResponseRedirect()` methods to render the HTML templates and redirect the user to a specific page. They also use the `get_object_or_404()` method to retrieve a specific object or raise a 404 error if it does not exist. The `@login_required` decorator ensures that the user is authenticated before accessing the views.