Subject

Django Documentation For A03 INET

Author: Mason Fraser
Date: 24/09/2019

Subject

Table of Contents

Subject

1. Project Overview

Complete the Django Tutorial Parts 1, 2, 3 and 4, install SQLite browser, include a J

2. Project Requirements

- Writing your first Django app, part 1 (show snippets)
- Writing your first Django app, part 2 (show snippets)
- Writing your first Django app, part 3 (show snippets)
- Writing your first Django app, part 4 (show snippets) SQLite3 DB Direct Access:

Install a SQLite Browser of your choice
Open the SQLite DB file used in your tutorial
Show the DB, and poll questions and answers

2.1. Requirements

Upload to your git repo (use proper directory structure)
Deploy these web pages/sites at the WEB-UB at 172.16.176.21
Upload a Formatted PDF (TOC, formal paragraphs, etc...) to D2L with screen snippets and links to your deployments. Use this document to formally describe how you think Django works, and how it's parts interact.

3. Tutorial Part 1 Polls App

Here I'll cover what I think is happening in this first tutorial.

3.1. Basic Configuration

The first thing needed is to ensure we have the correct version of Django and Python installed.

```
(venv) C:\Users\Mason Fraser\PycharmProjects\HelloWorld>python -m django --version
2.2.5

(venv) C:\Users\Mason Fraser\PycharmProjects\HelloWorld>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Pycharm creates a starting project for us, so we can skip to the next part.

3.2. views.py

Since this is a new application, we need a place to work in. So we create a new directory for the application, in this case “polls/”, This directory structure will house the poll application..

Now that we have a directory we need a view, which are a key component of the applications built with the framework. They take a python function, or class, that return a web response.

```
views.py   urls.py
from django.shortcuts import render
from django.http import HttpResponse


def index(request):
    return HttpResponse("Hello, Russ. You're at the polls index.")
# Create your views here.
```

This is about as simle of a view as we can create.

In order to call the view, we need to map it to a URLconf.

3.3. urls.py

In order to map the views to a url, we’ll have to create a urls.py file.

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

from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

We need to point the root URLconf at our new module to include our new app.

In HelloWorld/urls.py we need to import include() from django.urls

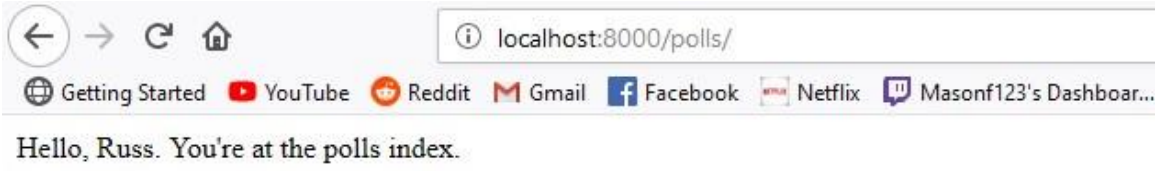
include() basically “roots” a set of URLs below other ones. This makes it easy to plug and play urls.

```
from django.contrib import admin
from django.urls import include, path

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

Once we’ve imported the include method, we point a path for the framework to find our url, so we add the line “path('polls/', include('polls.urls'))” which will pull the polls app if we go to the /polls/ url.

3.4. Links

At this point, the app should be up and running. So we run the framework and go to http://127.0.0.1:8000/polls/ for a local copy



4. Tutorial Part 2 Database Setup

Here I cover what’s done in the second tutorial

4.1. Database Initial Setup

We need to start by opening our settings.py, which is just module that represents Django settings. Since we’re in here we will set our timezone as well. America/Halifax

```
TIME_ZONE = 'America/Halifax'
```

Next we need to migrate, the migrate command looks at the INSTALLED_APPS and creates the necessasry tables according to our settings in settings.py

We enter “python manage.py mirgate” into our terminal window.

```
(venv) C:\Users\Mason Fraser\PycharmProjects\HelloWorld>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying sessions.0001_initial... OK
```

4.2. Creating Models

We’re using SQlite, since that’s the default.

In polls/models.py we need to create our models, Question, and Choice.

```python
from django.db import models

# Create your models here.


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)
```

Each model is represented by a class that subclasses “django.db.models.Model” .
Each model has a number of class variables

4.3. Activating Models

This part give Django a lot of information, which will let the framework create a database schema, and create a python database access api for accessing the objects above, but we need to tell polls the app is installed.

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    "polls.apps.PollsConfig",
]
```

Here I’ve added the dotted path to the installed apps. The framework now knows to include the polls app.

Now Django needs to be told we made the changes to the models with the following command.

“`python manage.py makemigrations polls`”

```
(venv) C:\Users\Mason Fraser\PycharmProjects\HelloWorld>python manage.py makemigrations polls
Migrations for 'polls':
  polls\migrations\0001_initial.py
    - Create model Question
    - Create model Choice
```

The changes are stored as a migration.

Now we just need to migrate to finalize the changes and apply them.

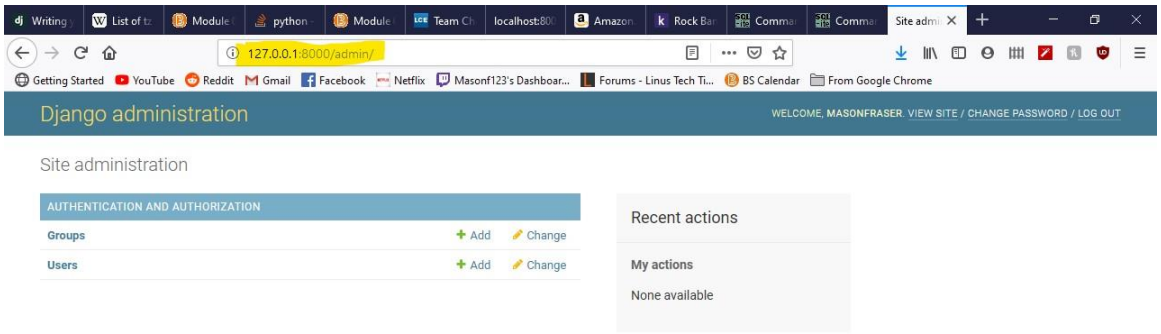
```
(venv) C:\Users\Mason Fraser\PycharmProjects\HelloWorld>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying polls.0001_initial... OK
```

4.4. Creating the SuperUser

We need to create a user who can login to the admin site. So we run the following command
"`python manage.py createsuperuser`"

```
(venv) C:\Users\Mason Fraser\PycharmProjects\HelloWorld>python manage.py createsuperuser
Username (leave blank to use 'masonfraser'):
Email address: masonjamesfraser@gmail.com
Password:
Password (again):
Superuser created successfully.
```

The Django admin site is activated by default, start the webserver and go to url /admin/



We don't see the poll app here, we need to tell the admin that Question objects have an admin interface.

```
from django.contrib import admin
from .models import Question
# Register your models here.

admin.site.register(Question)
```

Added the above to the polls/admin.py and refresh the webpage.

The admin can now play around with the questions.

5. Tutorial Part 3 Public Interface

A View is a webpage in your Django app that serves a particular function and template.

This poll application will have four views

- Index – This will display the latest questions
- Detail – This will display a question text, with no results but a place to vote
- Results – This will show he results for a question
- Vote Action – this handles voting for a particular choice in a particular question.

5.1. Writing more Views

These new views are slightly different, because they take an argument.

```
from django.http import HttpResponse


def index(request):
    return HttpResponse("Hello, Russ. You're at the polls index.")


def detail(request, question_id):
    return HttpResponse("You're looking at question %s." % question_id)


def results(request, question_id):
    response = "You're looking at the results of question %s."
    return HttpResponse(response % question_id)


def vote(request, question_id):
    return HttpResponse("You're voting on question %s." % question_id)
# Create your views here.
```

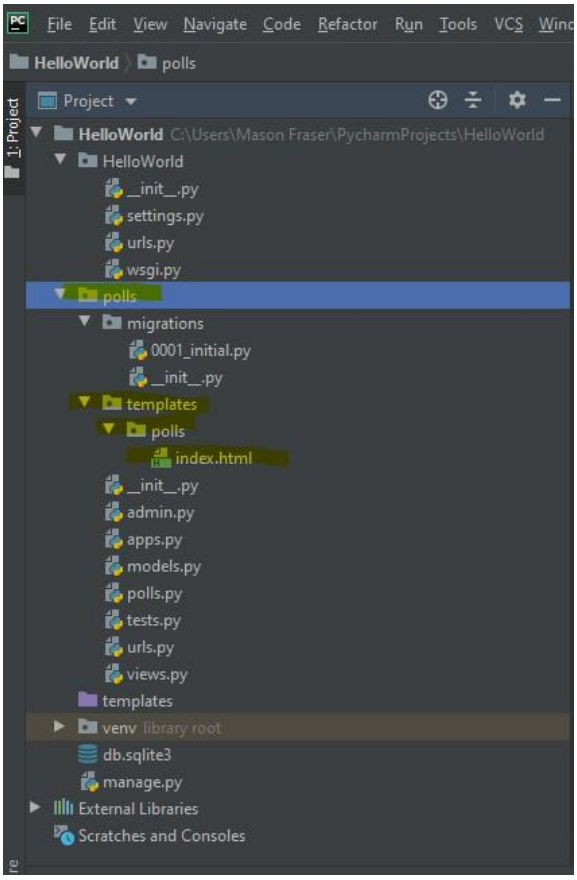
Now we have to add these new views into the URLconf, polls/urls.py

```
from django.urls import path

from . import views

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>/vote/', views.vote, name='vote'),
]
```

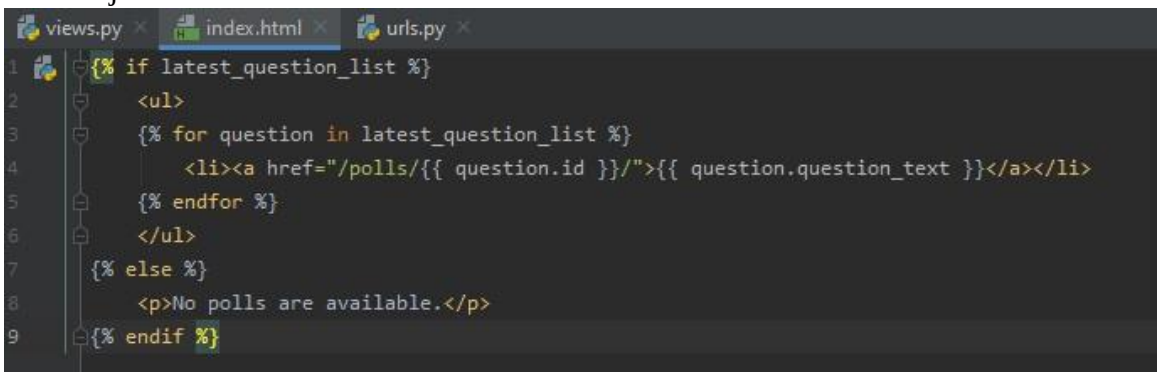
Each view is responsible for doing one of two things, either returning an HttpResponse object, or throwing a hissy fit and raising an exception such as 404.

5.2. Templates

Within the polls directory I've created /templates/ and within that I've created /polls/ and added an index.html file. Thanks to how the app_directories template loader works, we can refer to this simply as polls/index.html



Now we just add our code

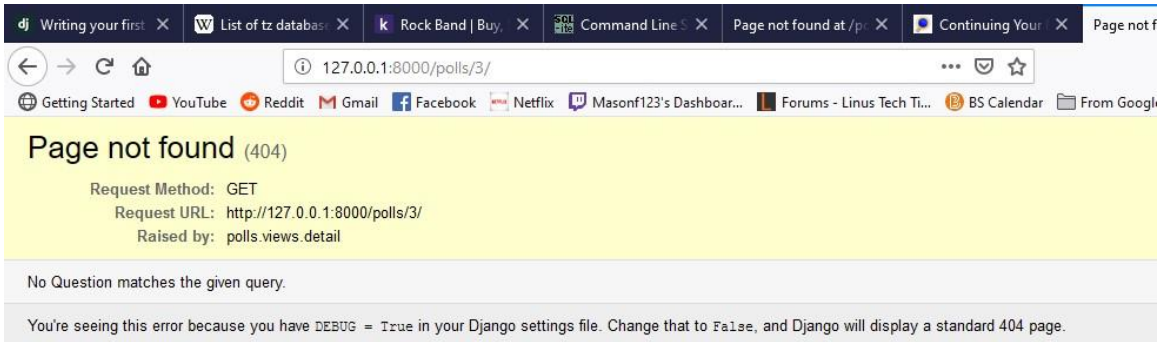


And update polls/views.py index method to acutally use the template.

```python
def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    template = loader.get_template('polls/index.html')
    context = {
        'latest_question_list': latest_question_list,
    }
    return HttpResponse(template.render(context, request))
```

5.3. Raising a 404 Error

We want the application to raise an error when a user attempts to view a question that doesn't exist.



The view raises the Http404 exception if a question with the requested ID doesn't exist.

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

5.4. The Template System

The template system uses dot-lookup syntax to access variable attributes, Django does a dictionary lookup on the object question, if that doesn't work it tries an attribute lookup.

6. Tutorial Part 4 Simple form Processing

Adding functionality to vote, process, and cut down on the code a bit.

6.1. Adding the results

I created a results.html in /polls/templates/polls/

```
<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>
```

And updated views.py

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

Now we can vote!

What's up?

- Not much -- 2 votes
- The sky -- 1 vote

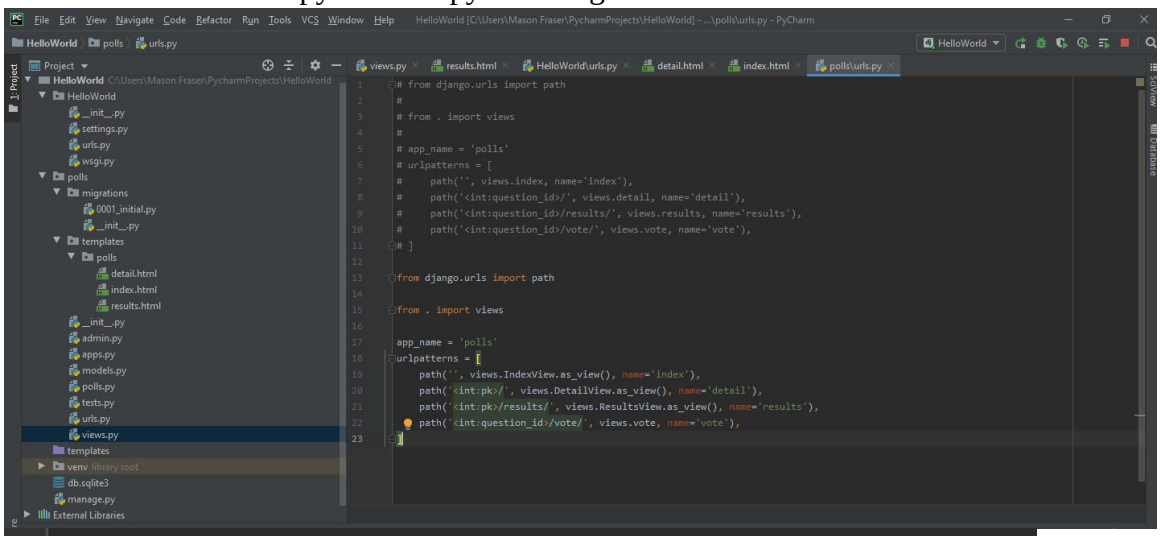
Vote again?

What's up?

◉ Not much
○ The sky
Vote

6.2. Cleanup the code

We amended the views.py and urls.py too use generic views



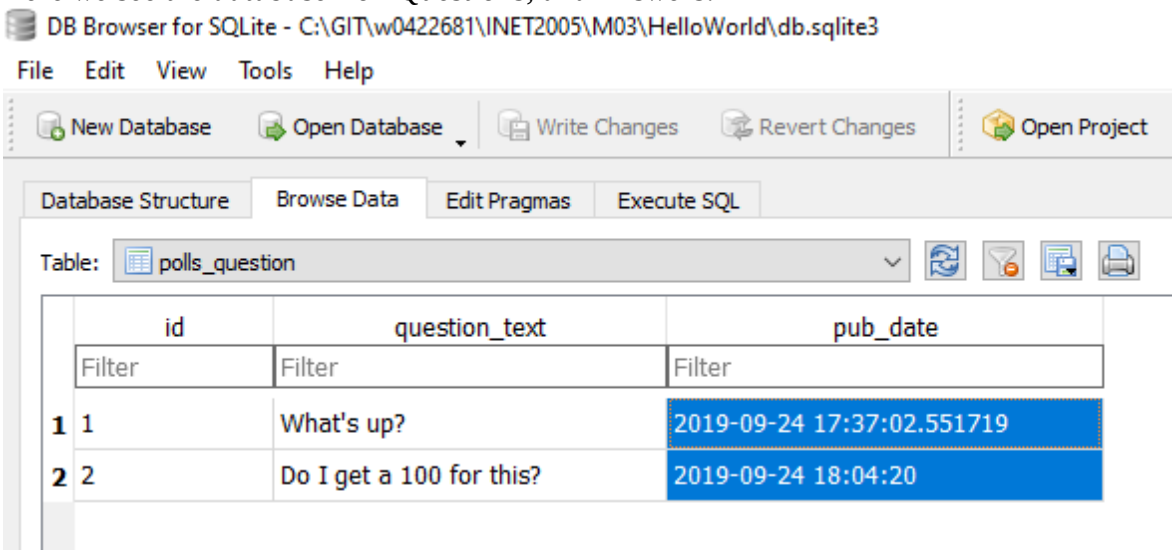

```
class IndexView(generic.ListView):
    template_name = 'polls/index.html'
    context_object_name = 'latest_question_list'

    def get_queryset(self):
        """Return the last five published questions."""
        return Question.objects.order_by('-pub_date')[:5]


class DetailView(generic.DetailView):
    model = Question
    template_name = 'polls/detail.html'


class ResultsView(generic.DetailView):
    model = Question
    template_name = 'polls/results.html'
```

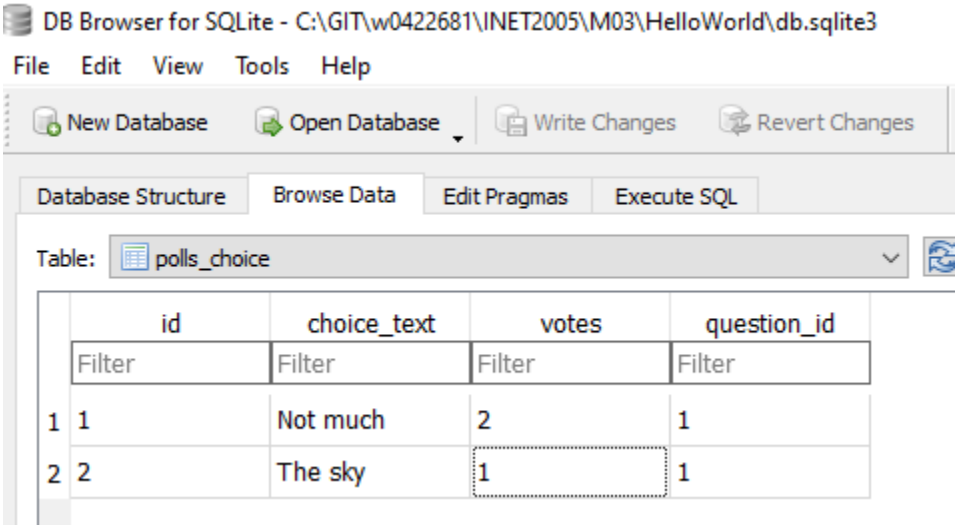
We use two generic views, ListView and DetailView.

Each view needs to know what model it will be actin on, we use the model attribute to provide this.

7. SQLite3 DB Direct Access

Here we see the database Poll Questions, and Answers.





8. A4 Django#2 Templates

The following is for the next assignment, we add a display a basic HTML table and JavaScript graph

8.1. HTML Table

This is the code for the table, it loops through the data in the choice_set and displays the data in a table.

```
{% for choice in question.choice_set.all %}
    <tr>
    <td>{{ question.question_text }} </td>
    <td>{{ choice.choice_text }} </td>
    <td>{{ choice.votes }}</td>
    </tr>
{% endfor %}
```

This is what the output looks like.

Question	Answer	# of Votes
What's up?	Not much	2
What's up?	The sky	2

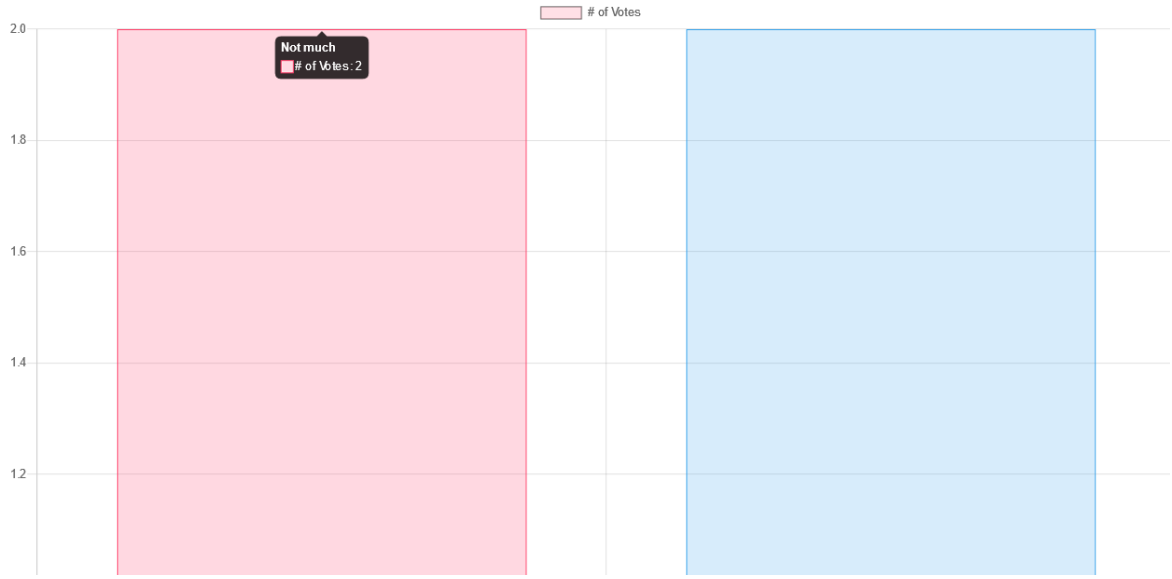
Vote again?

8.2. JavaScript Table

This is the code graph made with a module in JavaScript.

```
var ctx = document.getElementById('myChart').getContext('2d');
var myChart = new Chart(ctx, {
    type: 'bar',
    data: {
        labels: [{% for choice in question.choice_set.all %} "{{ choice.choice_text }}", {% endfor %}],
        datasets: [{
            label: '# of Votes',
            data: [{% for choice in question.choice_set.all %} {{ choice.votes }}, {% endfor %}],
            backgroundColor: [
                'rgba(255, 99, 132, 0.2)',
                'rgba(54, 162, 235, 0.2)',
                'rgba(255, 206, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(153, 102, 255, 0.2)',
                'rgba(255, 159, 64, 0.2)'
            ],
            borderColor: [
                'rgba(255, 99, 132, 1)',
                'rgba(54, 162, 235, 1)',
                'rgba(255, 206, 86, 1)',
                'rgba(75, 192, 192, 1)',
                'rgba(153, 102, 255, 1)',
                'rgba(255, 159, 64, 1)'
            ],
            borderWidth: 1
        }]
    },
    options: {
        scales: {
            yAxes: [{
                ticks: {
script › myChart › data › datasets › data
```

This graph takes out data like above, and loops through it to dynamically build and adjust the size as data changes. Every time a new choice gets added, or a vote is cast the graph will change and grow.



This is what it looks like on the webpage.