#Forms and Generic views

#example

```
<form action="{% url 'polls:vote' question.id %}" method="post">
        {% csrf_token %}
        <fieldset>
                <legend><h1>{{ question.question_text }}</h1></legend>
                {% if error_message %}
                        <p><strong>{{ error_message }}</strong></p>
                {% endif %}

                {% for choice in question.choice_set.all %}
                    <input type="radio" name="choice" id="choice{{ forloop.counter }}"
                value="{{ choice.id }}">
                     <label for="choice{{ forloop.counter }}">{{ choice.choice_text
                }}</label><br>
                {% endfor %}
        </fieldset>
        <input type="submit" value="Vote">
</form>
```

#explanation - basic concept of HTML forms

- display radio button each choice
- name of each radio button is "choice"
- value of each radio button is the associated question choice's ID

when somebody selects one of the radio button and submit it, it'll send POST data choice=#, where # is the ID of the selected choice

In the above example,

form's action is set to 'polls:vote' question.id and post method
forloop.counter - indicates how many times the for tag has gone through its loop
form is using POST method - {% csrf_token %} tag should be for protection

#in url file,

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

#in view file,

```
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):
        # Redisplay the question voting form.
        return render(request, 'polls/detail.html', {
            'question': question,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # Always return an HttpResponseRedirect after successfully dealing
        # with POST data. This prevents data from being posted twice if a
        # user hits the Back button.
        return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

request.POST - dictionary-like object lets to access submitted data by key name
request.POST['choice'] - returns ID of the selected choice, as a string

note: request.POST values are always strings and django also provides request.GET for accessing GET data in the same way

request.POST['choice'] - will raise KeyError if choice wasn't provided in POST data

After incrementing the choice count, the code returns an HttpResponseRedirect not HttpResponse

After somebody votes in a question, the vote() view redirects to the results page for the question. Let's write that view:

polls/views.py
```
from django.shortcuts import get_object_or_404, render

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

polls/results.html
```
<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>
```

Now, go to /polls/1/ in your browser and vote in the question.

It will navigate to the results page that gets updated each time you vote.
If the form is submitted without selecting a choice, it will throw an error message.

#generic views

- to avoid redundancy or common case
- by providing model attribute, we define which model is used in this view

two generic views used in above example: ListView and DetailView

ListView
- abstract the concepts of “display a list of objects”

DetailView
- abstract the concepts of “display a detail page for a particular type of object”
- expects the primary key value captured from the URL to be called "pk"
  so in example, question_id changed to pk for the generic views
- By default, template is called from <app name>/<model name>_detail.html
  in example, "polls/question_detail.html"

The template_name attribute is used to tell Django to use a specific template name instead of the auto generated default template name.

We also specify the template_name for the results list view – this ensures that the results view and the detail view have a different appearance when rendered, even though they’re both a DetailView behind the scenes.

Similarly, the ListView generic view uses a default template called <app name>/<model name>_list.html; we use template_name to tell ListView to use our existing "polls/index.html" template.

In previous parts of the tutorial, the templates have been provided with a context that contains the question and latest_question_list context variables.

For DetailView the question variable is provided automatically – since we’re using a Django model (Question), Django is able to determine an appropriate name for the context variable.

However, for ListView, the automatically generated context variable is question_list.
To override this we provide the context_object_name attribute, specifying that we want to use latest_question_list instead.

As an alternative approach, you could change your templates to match the new default context variables – but it’s a lot easier to tell Django to use the variable you want.

more info:

https://docs.djangoproject.com/en/4.1/topics/class-based-views/

Ref:
https://docs.djangoproject.com/en/4.1/intro/tutorial04/