## Напишите минимальную форму[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#write-a-minimal-form)

Давайте обновим наш шаблон сведений об опросе («polls/detail.html») с прошлого tutorial, чтобы шаблон содержал HTML-элемент:**<form>**

polls/templates/polls/detail.html[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#id1)

<**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**>

Краткое изложение:

* В приведенном выше шаблоне отображается переключатель для каждого варианта вопроса. Каждый переключатель является идентификатором соответствующего варианта вопроса. Каждый переключатель — . Это означает, что когда кто-то выбирает один из переключателей и отправляет форму, она отправит Данные POST, где # — идентификатор выбранного варианта. Это Основные понятия HTML-форм.**valuename"choice"choice=#**
* Мы устанавливаем форму на , и мы набор. Использование (в отличие от ) очень важно, потому что акт подачи этого Форма изменит данные на стороне сервера. Всякий раз, когда вы создаете форму, которая изменяется на стороне сервера данных, используйте . Этот совет не относится к конкретным Джанго; Это хорошая практика веб-разработки в целом.**action{% url 'polls:vote' question.id %}method="post"method="post"method="get"method="post"**
* **forloop.counter** указывает, сколько раз тег исчез через его петлю
* Поскольку мы создаем форму POST (которая может иметь эффект изменения data), нам нужно беспокоиться о подделках межсайтовых запросов. К счастью, вам не нужно слишком беспокоиться, потому что Django поставляется с Полезная система для защиты от него. Короче говоря, все формы POST, которые Ориентированные на внутренние URL-адреса, должны использовать тег шаблона.

Теперь давайте создадим представление Django, которое обрабатывает отправленные данные и делает что-то с этим. Помните, что в [уроке 3](https://docs.djangoproject.com/en/4.2/intro/tutorial03/) мы создал URLconf для приложения polls, который включает следующую строку:

polls/urls.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#id2)

path("<int:question\_id>/vote/", views.vote, name="vote"),

Мы также создали фиктивную реализацию функции. Давайте Создайте реальную версию. Добавить следующее:**vote()polls/views.py**

polls/views.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#id3)

**from** **django.http** **import** HttpResponse, HttpResponseRedirect

**from** **django.shortcuts** **import** get\_object\_or\_404, render

**from** **django.urls** **import** reverse

**from** **.models** **import** Choice, Question

*# ...*

**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,)))

Этот код включает в себя несколько вещей, которые мы еще не рассмотрели в этом руководстве:

* похож на словарь Объект, который позволяет получить доступ к отправленным данным по имени ключа. В этом случае возвращает идентификатор выбранного варианта в виде струна. Значения: всегда строки.**request.POST['choice']**Обратите внимание, что Django также предоставляет доступ к данным GET таким же образом – Но мы явно используем в нашем коде, чтобы гарантировать, что данные только изменено с помощью POST-вызова.
* **request.POST['choice']** поднимется, если не было указано в данных POST. Приведенный выше код проверяет и повторно отображает форму вопроса с ошибкой сообщение, если не дано.**choicechoice**
* После увеличения количества вариантов код возвращает атрибут, а не обычный . принимает один аргумент: URL-адрес, на который будет перенаправлен пользователь (см. следующий пункт о том, как это сделать) в этом случае мы конструируем URL).  
  Как указано в приведенном выше комментарии Python, вы всегда должны возвращать после успешной работы с Данные POST. Этот совет не относится к Django; Это хорошая веб-разработка Практика в целом.

В этом примере мы используем функцию в конструкторе. Эта функция помогает избежать жесткого кодирования URL-адреса в функции просмотра. Ему присваивается имя представления, которому мы хотим передать управление, и переменная часть шаблона URL-адреса, указывающая на это представление. В этом case, используя URLconf, который мы настроили в [Уроке 3](https://docs.djangoproject.com/en/4.2/intro/tutorial03/), Этот вызов вернет строку вида  
"/polls/3/results/"

* where the is the value of . This redirected URL will then call the view to display the final page.**3question.id'results'**

As mentioned in [Tutorial 3](https://docs.djangoproject.com/en/4.2/intro/tutorial03/), is an [**HttpRequest**](https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpRequest) object. For more on [**HttpRequest**](https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpRequest) objects, see the [request and response documentation](https://docs.djangoproject.com/en/4.2/ref/request-response/).**request**

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

polls/views.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#id4)

**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})

Это почти то же самое, что и вид из [урока 3](https://docs.djangoproject.com/en/4.2/intro/tutorial03/). Единственное отличие заключается в названии шаблона. Мы исправим это избыточность позже.**detail()**

Теперь создайте шаблон:**polls/results.html**

polls/templates/polls/results.html[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#id5)

<**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**>

Теперь зайдите в свой браузер и проголосуйте за вопрос. Вы должны увидеть страница результатов, которая обновляется каждый раз, когда вы голосуете. Если вы отправляете форму Не выбрав вариант, вы должны увидеть сообщение об ошибке.**/polls/1/**

**Заметка**

У кода для нашего представления есть небольшая проблема. Сначала он получает объект из базы данных, а затем вычисляет новый , а затем сохраняет его обратно в базу данных. Если два пользователя Ваш сайт пытается проголосовать *в одно и то же время*, это может пойти не так: То же значение, скажем, 42, будет получено для . Затем, для Оба пользователя вычисляют и сохраняют новое значение 43, но 44 будет Математическое ожидание.**vote()selected\_choicevotesvotes**

Это называется *состоянием гонки*. Если вам интересно, вы можете прочитать [Избегание условий гонки с помощью F(),](https://docs.djangoproject.com/en/4.2/ref/models/expressions/#avoiding-race-conditions-using-f) чтобы узнать, как вы можете решить эту проблему выпуск.

## Используйте универсальные представления: чем меньше кода, тем лучше[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#use-generic-views-less-code-is-better)

Представления (из [урока 3](https://docs.djangoproject.com/en/4.2/intro/tutorial03/)) очень короткие и, как упоминалось выше, избыточны. Представление, в котором отображается список опросов, аналогично.**detail()results()index()**

Эти представления представляют собой типичный случай базовой веб-разработки: получение данных из базу данных в соответствии с параметром, переданным в URL, загружая шаблон и возвращая визуализированный шаблон. Поскольку это так распространено, Django предоставляет ярлык, называемый системой «общих представлений».

Общие представления абстрагируют общие шаблоны до такой степени, что вам даже не нужно , чтобы написать код Python для написания приложения.

Давайте преобразуем наше приложение для опросов, чтобы использовать общую систему представлений, чтобы мы могли удалить куча собственного кода. Нам нужно будет сделать несколько шагов, чтобы сделать преобразование. Мы будем:

1. Преобразуйте URLconf.
2. Удалите некоторые старые, ненужные представления.
3. Вводите новые представления, основанные на общих представлениях Django.

Читайте дальше, чтобы узнать подробности.

**Почему перетасовка кода?**

Как правило, при написании приложения Django вы оцените, являются ли общие представления хорошо подходят для вашей проблемы, и вы будете использовать их с самого начала, вместо того, чтобы переделывать код на полпути. Но этот урок намеренно сосредоточился на написании взглядов «трудным путем» до сих пор, чтобы Сосредоточьтесь на основных концепциях.

Вы должны знать основы математики, прежде чем начать использовать калькулятор.

### **Изменить URLconf**[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#amend-urlconf)

Сначала откройте URLconf и измените его следующим образом:**polls/urls.py**

polls/urls.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#id6)

**from** **django.urls** **import** path

**from** **.** **import** views

app\_name = "polls"

urlpatterns = [

path("", views.IndexView.as\_view(), name="index"),

path("<int:pk>/", views.DetailView.as\_view(), name="detail"),

path("<int:pk>/results/", views.ResultsView.as\_view(), name="results"),

path("<int:question\_id>/vote/", views.vote, name="vote"),

]

Обратите внимание, что имя шаблона совпадает в строках пути второго и Третий паттерн изменился с на .**<question\_id><pk>**

### **Изменение представлений**[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#amend-views)

Далее мы собираемся удалить наши старые представления , и и вместо этого использовать общие представления Django. Для этого откройте файл и измените его следующим образом:**indexdetailresultspolls/views.py**

**polls/views.py**[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial04/#id7)

**from** **django.http** **import** HttpResponseRedirect

**from** **django.shortcuts** **import** get\_object\_or\_404, render

**from** **django.urls** **import** reverse

**from** **django.views** **import** generic

**from** **.models** **import** Choice, Question

**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"

**def** vote(request, question\_id):

... *# same as above, no changes needed.*

Здесь мы используем два общих представления: и . Соответственно, те Два представления абстрагируют понятия «отобразить список объектов» и «Отображение страницы сведений для определенного типа объекта».

* Каждое общее представление должно знать, по какой модели оно будет действовать на. Это обеспечивается с помощью атрибута.**model**
* Общее представление ожидает, что будет вызвано значение первичного ключа, полученное из URL-адреса, поэтому мы изменили его на generic Представления.**"pk"question\_idpk**

По умолчанию универсальный В представлении используется шаблон с именем . В нашем случае он будет использовать шаблон . Атрибут используется для того, чтобы указать Django использовать определенный Имя шаблона вместо автоматически созданного имени шаблона по умолчанию. Мы Также укажите для представления списка – Это гарантирует, что представление результатов и подробное представление будут иметь Разный внешний вид при рендеринге, хотя они оба находятся за кулисами.**<app name>/<model name>\_detail.html"polls/question\_detail.html"template\_nametemplate\_nameresults**

Точно так же общий view использует шаблон по умолчанию с именем ; Мы используем, чтобы сказать, чтобы использовать наш существующий шаблон.**<app name>/<model name>\_list.htmltemplate\_name"polls/index.html"**

В предыдущих частях учебника были предоставлены шаблоны с контекстом, содержащим переменные и context. Для переменной предусмотрен автоматически – так как мы используем модель Django (), Django может определить подходящее имя для контекстной переменной. Однако для ListView автоматически создаваемой контекстной переменной является . Чтобы переопределить это, мы предоставляем атрибут, указывая, что мы хотим использовать вместо него. В качестве альтернативного подхода вы можете изменить свои шаблоны, чтобы они соответствовали новые контекстные переменные по умолчанию - но гораздо проще сказать Django Используйте нужную переменную.**questionlatest\_question\_listDetailViewquestionQuestionquestion\_listcontext\_object\_namelatest\_question\_list**