## Обзор[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#overview)

Представление - это "тип" веб-страницы в вашем приложении Django, который обычно обслуживает конкретная функция и имеет определенный шаблон. Например, в блоге приложения, у вас могут быть следующие представления:

* Домашняя страница блога - отображает последние несколько записей.
* Страница «подробностей» записи – страница постоянной ссылки для одной записи.
* Страница архива на основе года – отображает все месяцы с записями в файле Данный год.
* Страница архива на основе месяцев – отображает все дни с записями в файле Задан месяц.
* Страница архива по дням – отображает все записи за данный день.
* Действие комментария — обрабатывает публикацию комментариев к данной записи.

В нашем приложении для опроса у нас будут следующие четыре представления:

* Страница «индекс» вопроса – отображает несколько последних вопросов.
* Страница «подробности» вопроса – отображает текст вопроса, без результатов, но с формой для голосования.
* Страница «результаты» вопроса – отображает результаты по конкретному вопросу.
* Действие голосования - обрабатывает голосование за определенный выбор в определенном вопрос.

В Django веб-страницы и другой контент доставляются представлениями. Каждое представление представлен функцией Python (или методом, в случае представлений на основе классов). Django выберет представление, изучив запрошенный URL-адрес (если быть точным, часть URL-адреса после доменного имени).

Теперь в свое время в Сети вы, возможно, сталкивались с такими красавицами, как . Вам будет приятно узнать, что Django позволяет нам гораздо более элегантные *шаблоны URL,* чем этот.**ME2/Sites/dirmod.htm?sid=&type=gen&mod=Core+Pages&gid=A6CD4967199A42D9B65B1B**

Шаблон URL-адреса — это общая форма URL-адреса, например: .**/newsarchive/<year>/<month>/**

Чтобы перейти от URL-адреса к представлению, Django использует так называемые 'URLconfs'. A URLconf сопоставляет шаблоны URL-адресов с представлениями.

В этом учебном пособии приведены основные инструкции по использованию URLconfs, и вы можете см. [Диспетчер URL-адресов](https://docs.djangoproject.com/en/4.2/topics/http/urls/) для получения дополнительной информации.

## Запись большего количества просмотров[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#writing-more-views)

Теперь давайте добавим еще несколько представлений в . Эти взгляды немного по-другому, потому что они берут аргумент:**polls/views.py**

опросы/просмотры.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id2)

**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)



Подключите эти новые представления к модулю, добавив следующие вызовы:**polls.urls**

polls/urls.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id3)

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

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

urlpatterns **=** [

*# ex: /polls/*

path("", views**.**index, name**=**"index"),

*# ex: /polls/5/*

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

*# ex: /polls/5/results/*

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

*# ex: /polls/5/vote/*

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

]

Загляните в свой браузер на «/polls/34/». Он запустит метод и отобразит любой идентификатор, который вы указали в URL-адресе. Попытка «/polls/34/results/» и «/polls/34/vote/» также отобразят Заполнители результатов и страницы голосования.**detail()**

Когда кто-то запрашивает страницу с вашего сайта - скажем, "/polls/34/", Django загрузит модуль Python, потому что на него указывает настройка. Он находит именованную переменную и просматривает шаблоны по порядку. Найдя совпадение в , он удаляет совпадающий текст () и отправляет оставшийся текст – – в URLconf 'polls.urls' для дальнейшей обработки. Вот оно совпадения , приводящий к вызову представления Вот так:**mysite.urlsurlpatterns'polls/'"polls/""34/"'<int:question\_id>/'detail()**

detail(request=<HttpRequest object>, question\_id=34)

Часть происходит от . Использование угла brackets «захватывает» часть URL-адреса и отправляет его в качестве аргумента ключевого слова в Функция просмотра. Часть строки определяет имя, которое будет использоваться для идентификации совпадающего шаблона, и деталь является converter, который определяет, какие шаблоны должны соответствовать этой части пути URL. Двоеточие () разделяет имя преобразователя и шаблона.**question\_id=34<int:question\_id>question\_idint:**

## Пишите представления, которые действительно что-то делают[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#write-views-that-actually-do-something)

Каждое представление отвечает за выполнение одного из двух действий: Возврат объекта, содержащего содержимое для запрошенная страница или создание исключения, например . Тем Остальное зависит от вас.

Представление может считывать записи из базы данных или нет. Он может использовать шаблон система, такая как Django - или сторонняя система шаблонов Python - или нет. Он может генерировать PDF-файл, выводить XML, создавать ZIP-файл на лету, что угодно вы хотите, используя любые библиотеки Python, которые вы хотите.

Все, чего хочет Django, это то, что . Или исключение.

Поскольку это удобно, давайте воспользуемся собственным API базы данных Django, который мы рассмотрели в [уроке 2](https://docs.djangoproject.com/en/4.2/intro/tutorial02/). Вот один удар по новому представлению, в котором отображаются последние 5 вопросов опроса в системе, разделенных Запятые, согласно дате публикации:**index()**

опросы/просмотры.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id4)

**from** **django.http** **import** HttpResponse

**from** **.models** **import** Question

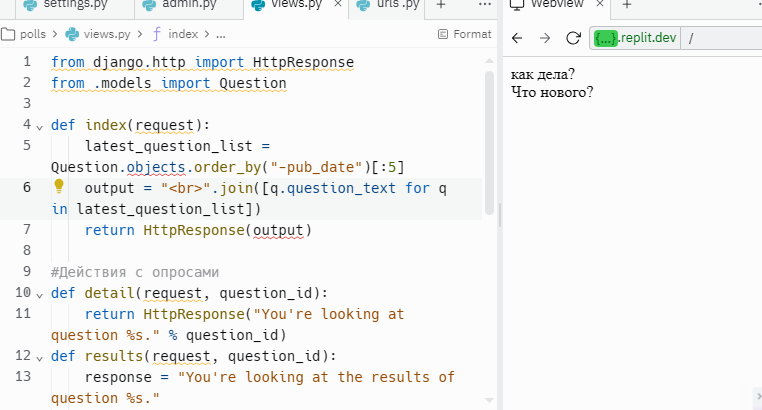
**def** index(request):

latest\_question\_list = Question.objects.order\_by("-pub\_date")[:5]

output = ", ".join([q.question\_text **for** q **in** latest\_question\_list])

**return** HttpResponse(output)

*# Leave the rest of the views (detail, results, vote) unchanged*



Однако здесь есть проблема: дизайн страницы жестко запрограммирован в представлении. Если Если вы хотите изменить внешний вид страницы, вам придется отредактировать этот код Python. Итак, давайте воспользуемся системой шаблонов Django, чтобы отделить дизайн от Python с помощью Создание шаблона, который может использовать представление.

Во-первых, создайте каталог с именем в вашем каталоге. Django будет искать там шаблоны.**templatespolls**

[Настройка вашего проекта](https://docs.djangoproject.com/en/4.2/ref/settings/#std-setting-TEMPLATES) описывает, как будет загружаться Django и Шаблоны рендеринга. Файл параметров по умолчанию настраивает серверную часть, для параметра которой задано значение . По соглашению ищет «шаблоны» в каждом из каталогов .**DjangoTemplatesTrueDjangoTemplates**

В каталоге, который вы только что создали, создайте еще один с именем , и в нем создайте файл с именем . Другими словами, ваш шаблон должен находиться в . Из-за того, как работает загрузчик шаблонов, как описано выше, вы можете ссылаться на этот шаблон в Django как .**templatespollsindex.htmlpolls/templates/polls/index.htmlapp\_directoriespolls/index.html**

**Интервал между именами шаблонов**

Теперь мы*, возможно*, сможем уйти от размещения наших шаблонов непосредственно (вместо того, чтобы создавать еще один подкаталог), Но на самом деле это была бы плохая идея. Django выберет первый шаблон Он определяет, чье имя совпадает, и был ли у вас шаблон с таким же именем в *другом* приложении Django не смог бы различать им. Мы должны быть в состоянии указать Django на правильный и лучший Чтобы убедиться в этом, *расставьте их по именам*. То есть, поставив те templates в *другом* каталоге, названном в честь самого приложения.**polls/templatespolls**

Поместите в этот шаблон следующий код:

polls/templates/polls/index.html[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id5)

{% **if** latest\_question\_list %}

<**ul**>

{% **for** question **in** latest\_question\_list %}

<**li**><**a** href="/polls/{{ question.id }}/">{{ question.question\_text }}</**a**></**li**>

{% **endfor** %}

</**ul**>

{% **else** %}

<**p**>No polls are available.</**p**>

{% **endif** %}

**Заметка**

Чтобы сделать учебник короче, во всех примерах шаблонов используется неполный HTML. В В ваших собственных проектах вы должны использовать [полные HTML-документы](https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started#anatomy_of_an_html_document).

Теперь давайте обновим наше представление, чтобы использовать шаблон **index: polls/views.py**

**from** **django.http** **import** HttpResponse

**from** **django.template** **import** loader

**from** **.models** **import** Question

**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))

Этот код загружает вызванный шаблон и передает ему контекст. Контекст представляет собой словарь, сопоставляющий имена переменных шаблона с Python Объектов.**polls/index.html**

Загрузите страницу, указав в браузере на «/polls/», и вы должны увидеть маркированный список, содержащий вопрос "What's up" из [урока 2](https://docs.djangoproject.com/en/4.2/intro/tutorial02/). Ссылка указывает на страницу с подробными сведениями о вопросе.

### **Ярлык:**[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#a-shortcut-render)

Это очень распространенная идиома для загрузки шаблона, заполнения контекста и возврата объекта с результатом рендеринга шаблон. Django предоставляет ярлык. Вот полный вид, Переписать:**index()**

polls/views.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id7)

**from** **django.shortcuts** **import** render

**from** **.models** **import** Question

**def** index(request):

latest\_question\_list = Question.objects.order\_by("-pub\_date")[:5]

context = {"latest\_question\_list": latest\_question\_list}

**return** render(request, "polls/index.html", context)

Обратите внимание, что как только мы сделаем это во всех этих представлениях, нам больше не нужно импортировать и (вы будете хотите сохранить, если у вас все еще есть методы-заглушки для , и ).**HttpResponsedetailresultsvote**

Функция принимает объект запроса в качестве своего первый аргумент, имя шаблона в качестве второго аргумента и словарь в качестве его необязательный третий аргумент. Он возвращает объект заданного шаблона, отображаемый с заданным контекстом.

## Возникновение ошибки 404[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#raising-a-404-error)

Теперь давайте займемся подробным представлением вопроса - страницей, на которой отображается текст вопроса для данного опроса. Вот вид:

polls/views.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id8)

**from** **django.http** **import** Http404

**from** **django.shortcuts** **import** render

**from** **.models** **import** Question

*# ...*

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

**try**:

question = Question.objects.get(pk=question\_id)

**except** Question.DoesNotExist:

**raise** Http404("Question does not exist")

**return** render(request, "polls/detail.html", {"question": question})

Новая концепция: взгляд поднимает исключение если вопрос с запрошенным идентификатором не существует.

Мы немного обсудим, что вы могли бы поместить в этот шаблон позже, но если вы хотите быстро заставить работать приведенный выше пример, файл содержащий только:**polls/detail.html**

polls/templates/polls/detail.html[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id9)

{{ question }}

поможет вам начать прямо сейчас.

### **Ярлык:**[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#a-shortcut-get-object-or-404)

Это очень распространенная идиома, которую можно использовать и поднимать, если объект не существует. Джанго Предоставляет ярлык. Вот вид, переписанный:**detail()**

опросы/просмотры.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id10)

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

**from** **.models** **import** Question

*# ...*

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

question = get\_object\_or\_404(Question, pk=question\_id)

**return** render(request, "polls/detail.html", {"question": question})

Функция принимает модель Django в качестве первого аргумента и произвольного количества аргументов ключевых слов, которые он переходит к функции Менеджер модели. Он возникает, если объект этого не делает существовать.

**Философия**

Почему мы используем вспомогательную функцию вместо того, чтобы автоматически перехватывать исключения на более высоком уровне level или повышение API модели вместо ?

Потому что это свяжет слой модели со слоем представления. Один из Основные цели дизайна Django - поддерживать слабую связь. Несколько В модуле введена управляемая муфта.

Также есть функция, которая работает Точно так же, как - за исключением использования вместо . Он возникает, если список пуст.

## Используйте систему шаблонов[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#use-the-template-system)

Вернемся к представлению нашего приложения для опроса. Учитывая контекст переменная , вот как может выглядеть шаблон любить:**detail()questionpolls/detail.html**

polls/templates/polls/detail.html[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id11)

<**h1**>{{ question.question\_text }}</**h1**>

<**ul**>

{% **for** choice **in** question.choice\_set.all %}

<**li**>{{ choice.choice\_text }}</**li**>

{% **endfor** %}

</**ul**>

The template system uses dot-lookup syntax to access variable attributes. In the example of , first Django does a dictionary lookup on the object . Failing that, it tries an attribute lookup – which works, in this case. If attribute lookup had failed, it would’ve tried a list-index lookup.**{{ question.question\_text }}question**

Method-calling happens in the [**{% for %}**](https://docs.djangoproject.com/en/4.2/ref/templates/builtins/#std-templatetag-for) loop: is interpreted as the Python code , which returns an iterable of objects and is suitable for use in the [**{% for %}**](https://docs.djangoproject.com/en/4.2/ref/templates/builtins/#std-templatetag-for) tag.**question.choice\_set.allquestion.choice\_set.all()Choice**

See the [template guide](https://docs.djangoproject.com/en/4.2/topics/templates/) for more about templates.

## Removing hardcoded URLs in templates[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#removing-hardcoded-urls-in-templates)

Remember, when we wrote the link to a question in the template, the link was partially hardcoded like this:**polls/index.html**

<**li**><**a** href="/polls/{{ question.id }}/">{{ question.question\_text }}</**a**></**li**>

The problem with this hardcoded, tightly-coupled approach is that it becomes challenging to change URLs on projects with a lot of templates. However, since you defined the name argument in the [**path()**](https://docs.djangoproject.com/en/4.2/ref/urls/#django.urls.path) functions in the module, you can remove a reliance on specific URL paths defined in your url configurations by using the template tag:**polls.urls{% url %}**

<**li**><**a** href="{% **url** 'detail' question.id %}">{{ question.question\_text }}</**a**></**li**>

Это работает следующим образом: поиск определения URL-адреса, указанного в модуле. Вы можете точно увидеть, где находится URL-имя «detail» Ниже приведено определение:**polls.urls**

...

*# the 'name' value as called by the {% url %} template tag*

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

...

Если вы хотите изменить URL-адрес подробного представления опросов на что-то другое, Возможно, к чему-то вроде того, что вместо того, чтобы делать это в шаблон (или шаблоны), в котором вы бы его изменили:**polls/specifics/12/polls/urls.py**

...

*# added the word 'specifics'*

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

...

## Интервал между именами URL-адресов[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#namespacing-url-names)

В учебном проекте есть только одно приложение, . В реальных проектах Django Приложений может быть пять, десять, двадцать или больше. Как Django дифференцируется URL-имена между ними? Например, у приложения есть представление, как и у приложения в том же проекте, что и для блога. Как можно сделать так, чтобы Django знал, какое представление приложения создать для url при использовании тега template?**pollspollsdetail{% url %}**

Ответ заключается в добавлении пространств имен в URLconf. В файле добавьте его, чтобы задать пространство имен приложения:**polls/urls.pyapp\_name**

polls/urls.py[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id12)

**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>/vote/", views.vote, name="vote"),

]

Теперь измените свой шаблон с:**polls/index.html**

polls/templates/polls/index.html[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id13)

<**li**><**a** href="{% **url** 'detail' question.id %}">{{ question.question\_text }}</**a**></**li**>

Чтобы указать на подробное представление пространства имен, выполните следующие действия.

polls/templates/polls/index.html[**¶**](https://docs.djangoproject.com/en/4.2/intro/tutorial03/#id14)

<**li**><**a** href="{% **url** 'polls:detail' question.id %}">{{ question.question\_text }}</**a**></**li**>