Creating a project**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial01/#creating-a-project)**

If this is your first time using Django, you’ll have to take care of some initial setup. Namely, you’ll need to auto-generate some code that establishes a Django [project](https://docs.djangoproject.com/en/2.2/glossary/#term-project) – a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings.

From the command line, **cd** into a directory where you’d like to store your code, then run the following command:

/ 

**$** django-admin startproject mysite

The development server**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial01/#the-development-server)**

Let’s verify your Django project works. Change into the outer **mysite** directory, if you haven’t already, and run the following commands:

/ 

**$** python manage.py runserver

Creating the Polls app**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial01/#creating-the-polls-app)**

Now that your environment – a “project” – is set up, you’re set to start doing work.

Each application you write in Django consists of a Python package that follows a certain convention. Django comes with a utility that automatically generates the basic directory structure of an app, so you can focus on writing code rather than creating directories.

**Projects vs. apps**

What’s the difference between a project and an app? An app is a Web application that does something – e.g., a Weblog system, a database of public records or a simple poll app. A project is a collection of configuration and apps for a particular website. A project can contain multiple apps. An app can be in multiple projects.

Your apps can live anywhere on your [Python path](https://docs.python.org/3/tutorial/modules.html#tut-searchpath). In this tutorial, we’ll create our poll app right next to your **manage.py** file so that it can be imported as its own top-level module, rather than a submodule of **mysite**.

To create your app, make sure you’re in the same directory as **manage.py** and type this command:

/ 

**$** python manage.py startapp polls

Write your first view**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial01/#write-your-first-view)**

Let’s write the first view. Open the file **polls/views.py** and put the following Python code in it:

polls/views.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial01/#id1)**

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

**def** index(request):

**return** HttpResponse("Hello, world. You're at the polls index.")

In the **polls/urls.py** file include the following code:

polls/urls.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial01/#id2)**

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

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

urlpatterns = [

path('', views.index, name='index'),

]

The next step is to point the root URLconf at the **polls.urls** module. In **mysite/urls.py**, add an import for **django.urls.include** and insert an **[include()](https://docs.djangoproject.com/en/2.2/ref/urls/#django.urls.include)** in the **urlpatterns** list, so you have:

mysite/urls.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial01/#id3)**

**from** **django.contrib** **import** admin

**from** **django.urls** **import** include, path

urlpatterns = [

path('polls/', include('polls.urls')),

path('admin/', admin.site.urls),

]

python manage.py migrate

Creating models**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial02/#creating-models)**

In our simple poll app, we’ll create two models: **Question** and **Choice**. A **Question** has a question and a publication date. A **Choice** has two fields: the text of the choice and a vote tally. Each **Choice** is associated with a **Question**.

These concepts are represented by simple Python classes. Edit the **polls/models.py** file so it looks like this:

polls/models.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial02/#id2)**

**from** **django.db** **import** models

**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)

mysite/settings.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial02/#id3)**

INSTALLED\_APPS = [

'polls.apps.PollsConfig',

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

**$** python manage.py makemigrations polls

**Creating an admin user[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial02/#creating-an-admin-user)**

First we’ll need to create a user who can login to the admin site. Run the following command:

/ 

**$** python manage.py createsuperuser

**Make the poll app modifiable in the admin[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial02/#make-the-poll-app-modifiable-in-the-admin)**

But where’s our poll app? It’s not displayed on the admin index page.

Just one thing to do: we need to tell the admin that **Question** objects have an admin interface. To do this, open the **polls/admin.py** file, and edit it to look like this:

polls/admin.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial02/#id6)**

**from** **django.contrib** **import** admin

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

admin.site.register(Question)

Writing more views**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#writing-more-views)**

Now let’s add a few more views to **polls/views.py**. These views are slightly different, because they take an argument:

polls/views.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#id1)**

**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)

Wire these new views into the **polls.urls** module by adding the following **[path()](https://docs.djangoproject.com/en/2.2/ref/urls/#django.urls.path)** calls:

polls/urls.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#id2)**

**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'),

]

Put the following code in that template:

polls/templates/polls/index.html**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#id4)**

{% **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** %}

**A shortcut: [render()](https://docs.djangoproject.com/en/2.2/topics/http/shortcuts/#django.shortcuts.render)[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#a-shortcut-render)**

It’s a very common idiom to load a template, fill a context and return an **[HttpResponse](https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.HttpResponse)** object with the result of the rendered template. Django provides a shortcut. Here’s the full **index()** view, rewritten:

polls/views.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#id6)**

**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)

**A shortcut: [get\_object\_or\_404()](https://docs.djangoproject.com/en/2.2/topics/http/shortcuts/#django.shortcuts.get_object_or_404)[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#a-shortcut-get-object-or-404)**

It’s a very common idiom to use **[get()](https://docs.djangoproject.com/en/2.2/ref/models/querysets/#django.db.models.query.QuerySet.get)** and raise **[Http404](https://docs.djangoproject.com/en/2.2/topics/http/views/#django.http.Http404)** if the object doesn’t exist. Django provides a shortcut. Here’s the **detail()** view, rewritten:

polls/views.py**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#id9)**

**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})

Use the template system**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#use-the-template-system)**

Back to the **detail()** view for our poll application. Given the context variable **question**, here’s what the **polls/detail.html**template might look like:

polls/templates/polls/detail.html**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#id10)**

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

<**ul**>

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

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

{% **endfor** %}

</**ul**>

Removing hardcoded URLs in templates**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial03/#removing-hardcoded-urls-in-templates)**

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

<**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/2.2/ref/urls/#django.urls.path)** functions in the **polls.urls** module, you can remove a reliance on specific URL paths defined in your url configurations by using the **{% url %}** template tag:

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

app\_name = 'polls'

Write a simple form**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial04/#write-a-simple-form)**

Let’s update our poll detail template (“polls/detail.html”) from the last tutorial, so that the template contains an HTML **<form>**element:

polls/templates/polls/detail.html**[¶](https://docs.djangoproject.com/en/2.2/intro/tutorial04/#id1)**

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

{% **if** error\_message %}<**p**><**strong**>{{ error\_message }}</**strong**></**p**>{% **endif** %}

<**form** action="{% **url** 'polls:vote' question.id %}" method="post">

{% **csrf\_token** %}

{% **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** %}

<**input** type="submit" value="Vote">

</**form**>

We also created a dummy implementation of the **vote()** function. Let’s create a real version. Add the following to **polls/views.py**:

polls/views.py**[¶](https://docs.djangoproject.com/en/2.2/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,)))

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**[¶](https://docs.djangoproject.com/en/2.2/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})

Now, create a **polls/results.html** template:

polls/templates/polls/results.html**[¶](https://docs.djangoproject.com/en/2.2/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**>