Creating and Running Your First Django Project

This feature is only supported in the Professional edition.

Before you start

Make sure that the following prerequisites are met:

You are working with PyCharm version 2016.1 or higher. If you still do not have PyCharm, download it from [this page](https://www.jetbrains.com/pycharm/download/). To install PyCharm, follow the instructions, depending on your platform.

You have at least one Python interpreter properly installed on your computer. You can download an interpreter from [this page](http://python.org/download/).

You have Django package installed. To learn how to install packages using the PyCharm UI, read the section [Installing, Uninstalling and Upgrading Packages](https://www.jetbrains.com/help/pycharm/installing-uninstalling-and-upgrading-packages.html). You can also install Django as described in the page [How to install Django](https://docs.djangoproject.com/en/1.8/topics/install/).

This tutorial has been created with the following assumptions:

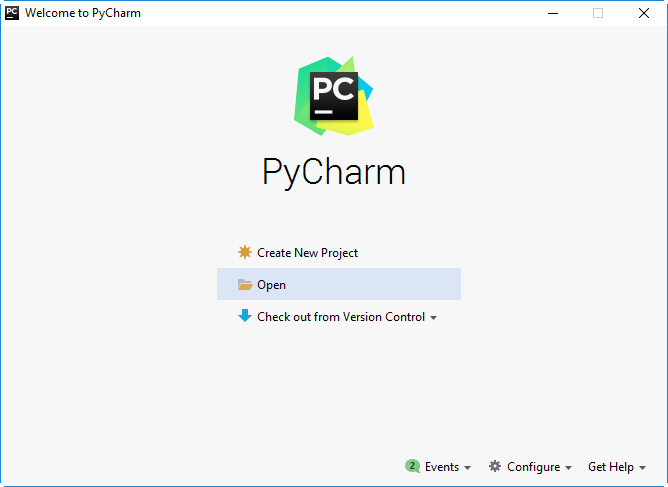
Python 3.4.1.

Django 1.10.0 or higher.

The example used in this tutorial is similar to the one used in [Django documentation](https://docs.djangoproject.com/en/1.8/). Sample project can be downloaded from [here](https://github.com/yzozulya/djtp_first_steps).

Creating a new project

Actually, all new projects are created same way: by clicking the Create New Project button in the Quick Start area of the [Welcome screen](https://www.jetbrains.com/help/pycharm/welcome-screen.html):

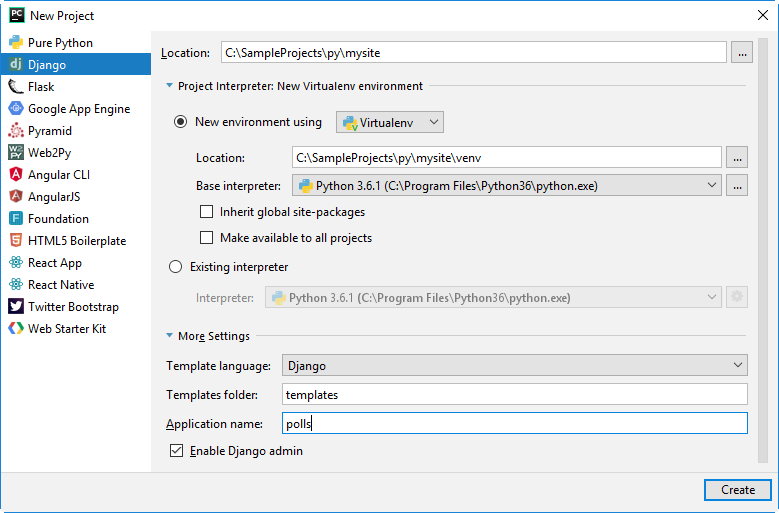


If you have an already opened project, create a new one by choosing File | New Project... on the main menu.

Then, select the desired project type (here it is Django). Specify the project name and location.

Python best practice is to create a virtualenv for each project. To do that, expand the Project Interpreter: New Virtualenv Environment node and select a tool used to create a new vitrual environment. Let's choose Virtualenv tool, and specify the location and base interpreter used for the new virtual environment. Select the two check boxes below if necessary.

.  
Next, expand the More Settings node and specify the Django-related settings. In the Application name field specify the application name (here it is polls).



Click Create - the Django project is ready.

Exploring project structure

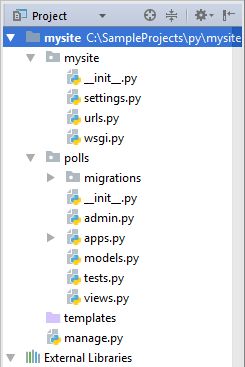
As mentioned above, basically, the stub project is ready. It contains framework-specific files and directories. Same happens when you create a project of any supported type, be it Pyramid, or Google App Engine.

Let's see how the structure of the new project is visible in the [Project Tool Window](https://www.jetbrains.com/help/pycharm/project-tool-window.html).

Project view of the Project tool window

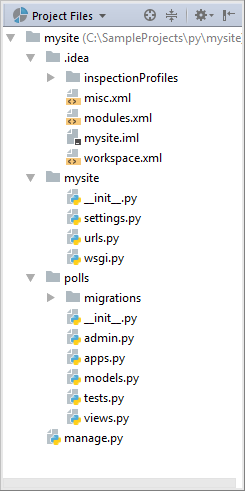
This view is displayed by default. It shows the Django-specific project structure: polls and mysite directories; also, you see the manage.py and settings.py files.

Note that you cannot see the .idea directory in this view:



Project Files view of the Project tool window

If for some reasons you would like to see contents of the .idea directory, choose the view Project Files: as you see, this view shows same directories and files, plus .idea directory, since is located under the project root:



Let's return to the Project view.

What do we see in the Project view?

mysite directory is a container for your project. In the Project view it is denoted with bold font.

manage.py: This is a command-line utility that lets you interact with your Django project. Refer to the [Django documentation](http://django.readthedocs.org/en/latest/ref/django-admin.html) for details.

The nested directory mysite is the actual Python package for your project.

mysite/\_\_init\_\_.py: This empty file tells Python that this directory should be considered a Python package.

mysite/settings.py: This file contains [configuration for your Django project](http://django.readthedocs.org/en/latest/topics/settings.html).

mysite/urls.py: This file contains the [URL declarations for your Django project](http://django.readthedocs.org/en/latest/topics/http/urls.html).

mysite/wsgi.py: This file defines an entry-point for WSGI-compatible web servers to serve your project. See [How to deploy with WSGI](http://django.readthedocs.org/en/latest/howto/deployment/wsgi/index.html) for more details.

The nested directory polls contains all the files required for developing a Django application (at this moment, these files are empty):

Again, polls/\_init\_.py tells Python that this directory should be considered a Python package.

polls/models.py: In this file, we'll [create models](https://www.jetbrains.com/help/pycharm/step-4-creating-and-running-your-first-django-project.html#creating_models) for our application.

polls/views.py: In this file, we'll [create views](https://www.jetbrains.com/help/pycharm/step-4-creating-and-running-your-first-django-project.html#creating_views).

templates directory is by now empty. It should contain the template files.

The nested directory migrations contains by now only the package file \_init\_.py, but will be used in the future to propagate the changes you make to your models (adding a field, deleting a model, etc.) into your database schema. Read the migrations description [here](https://docs.djangoproject.com/en/1.8/topics/migrations/).

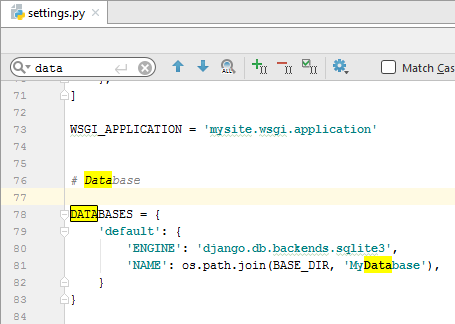
Note that you can create as many Django applications as needed. To add an application to a project, run the startapp task of the manage.py utility (Tools | Run manage.py task, then type startapp in the console).

Configuring the database

Now, when the project stub is ready, let's do some fine tuning. Open for editing settings.py. To do it, select the file in the Project tool window, and press F4. The file opens in its own tab in the editor.

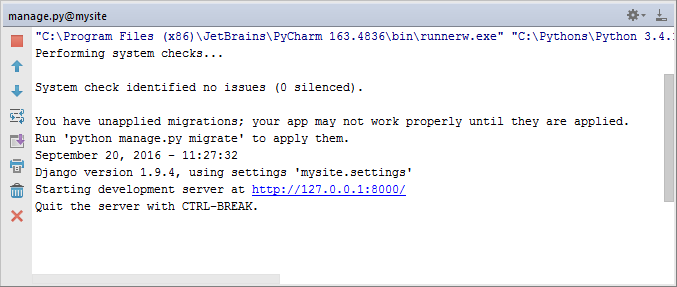
Specify which database you are going to use in your application. For this purpose, find the DATABASES variable: click Ctrl+F, and in the search field start typing the string you are looking for. Then, in the 'ENGINE' line, add the name of your database management system after dot (you can use any one specified after comment, but for the beginning we'll start with sqlite3.)

In the 'NAME' line, enter the name of the desired database, even though it doesn't yet exist.

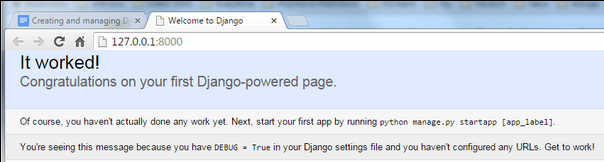


Launching Django server

Since we've prudently chosen sqlite3, we don't need to define the other values (user credentials, port and host). Let's now check whether our settings are correct. This can be done most easily: just launch the runserver task of the manage.py utility: press Ctrl+Alt+R, and enter task name in the manage.py console:



Follow the suggested link and see the following page:



Creating models

Next, open for editing the file models.py, and note that import statement is already there. Then type the following code:

from django.db import models

# the following lines added:

import datetime

from django.utils import timezone

class Question(models.Model):

question\_text = models.CharField(max\_length=200)

pub\_date = models.DateTimeField('date published')

def \_\_str\_\_(self):

return self.question\_text

def was\_published\_recently(self):

now = timezone.now()

return now - datetime.timedelta(days=1) <= self.pub\_date <= now

was\_published\_recently.admin\_order\_field = 'pub\_date'

was\_published\_recently.boolean = True

was\_published\_recently.short\_description = 'Published recently?'

class Choice(models.Model):

question = models.ForeignKey(Question)

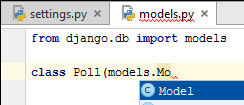
choice\_test = models.CharField(max\_length=200)

votes = models.IntegerField(default=0)

def \_\_str\_\_(self):

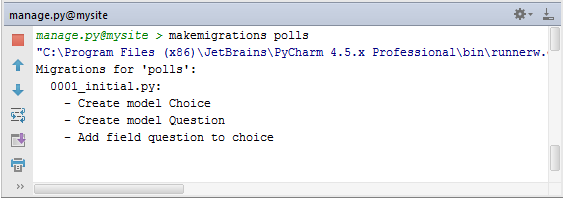
return self.choice\_test

Actually, you can just copy-paste, but typing is advisable - it helps you see the powerful PyCharm's code completion in action:

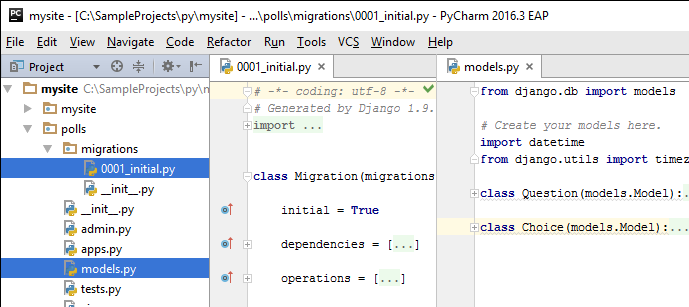


Creating database

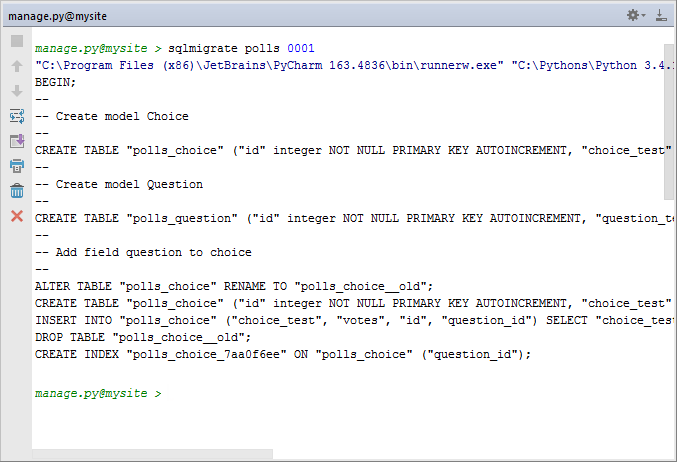
We have to create tables for the new model. For this purpose, we'll use the magic Ctrl+Alt+R shortcut to invoke the manage.py console. First command to perform is makemigrations polls:



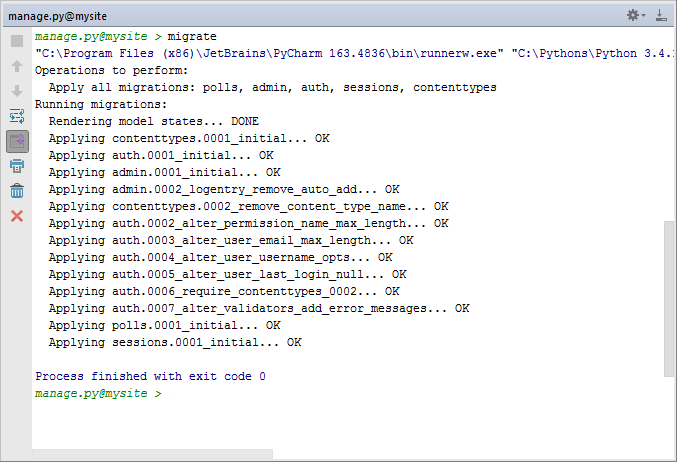
Thus you’ve told Django that two new models have been created, namely, Choice and Question, and created a migration:



Next, after the prompt, type the following command:

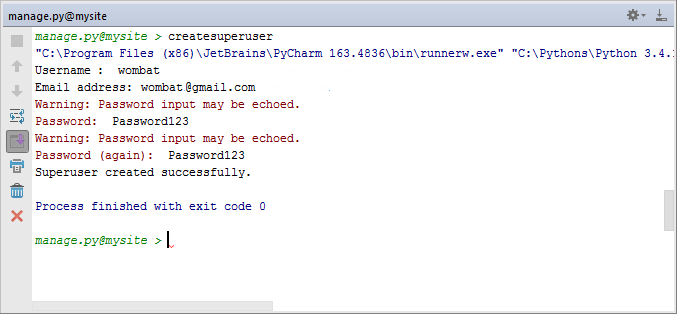
sqlmigrate polls 0001 

Finally, run migrate command to actually create these tables in your database:



Performing administrative functions

First thing, create a superuser. To do that, type the createsuperuser command in the manage.py console, specify your email address, and password:



Since we've decided to enable site administration, PyCharm has already uncommented the corresponding lines in the file urls.py.

Open the admin.py file in the polls directory for editing, and see the following code that already exists:

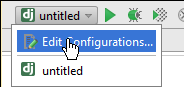
from django.contrib import admin

However, we need to enable editing functionality for the admin site.

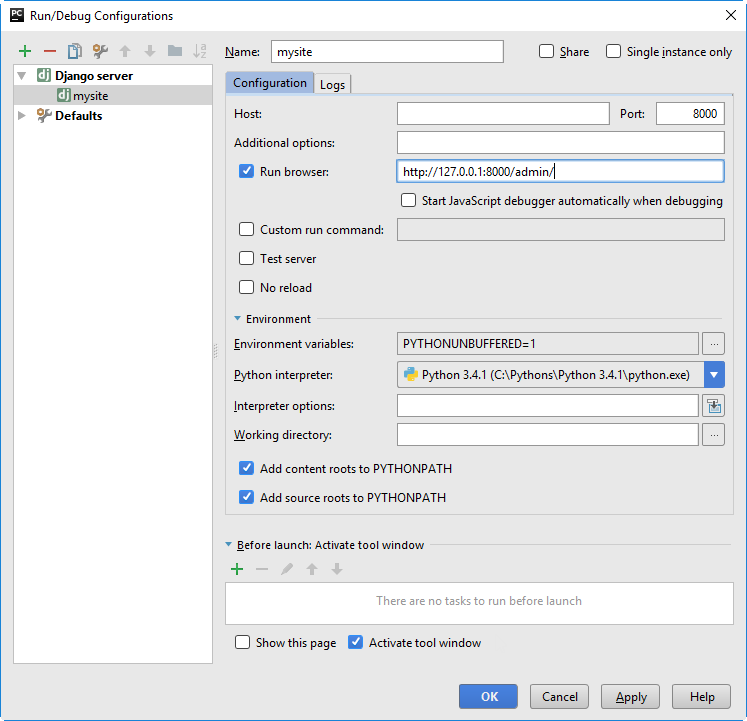
Preparing run/debug configuration

We are now ready to go to the admin page. Sure, it is quite possible to run the Django server, then go to your browser, and type the entire URL in the address bar, but with PyCharm there is an easier way: use the pre-configured Django server run configuration with some slight modifications.

To open this run/debug configuration for editing, on the main toolbar, click the run/debug configurations selector, and then choose Edit Configurations (or choose Run | Edit Configurations on the main menu):

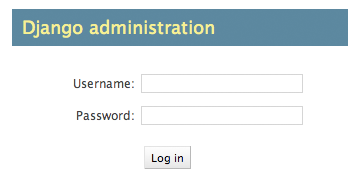


In the [Run/Dug Configuration dialog box](https://www.jetbrains.com/help/pycharm/run-debug-configuration-django-server.html), give this run/debug configuration a name (here it is mysite), enable running the application in the default browser (select the checkbox Run browser) and specify the page of the site to be opened by default (here this page is http://127.0.0.1:8000/admin/):



Launching the admin site

Now, to launch the application, press Shift+F10, or click runon the main toolbar to open the standard Django site login page:



After you log in, the administration page is displayed. It has a section Authentication and Authorization (Groups and Users), but Polls is not available. Why so?

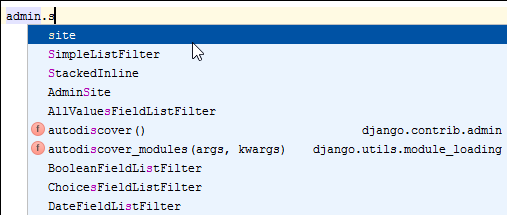
We must tell admin that Question objects have an admin interface; to do that, let’s open the file polls/admin.py for editing (select it in Project view and press F4), and type the following code:

from django.contrib import admin

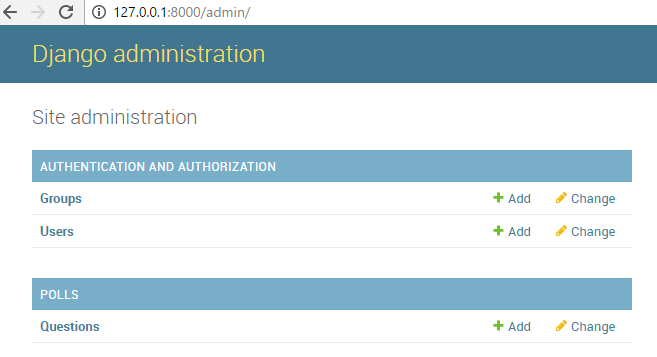
from .models import Question #this line added

admin.site.register(Question)#this line added

Again pay attention to the code completion:



Refresh the page and see that Polls section with Questions appeared:



Click Add to create some questions.

Editing admin.py

However, each question has a number of choices, but choices are still not available. Again, open for editing the file polls/admin.py and change it as follows:

from django.contrib import admin

from .models import Choice, Question

class ChoiceInline(admin.TabularInline):

model = Choice

extra = 3

class QuestionAdmin(admin.ModelAdmin):

fieldsets = [

(None, {'fields': ['question\_text']}),

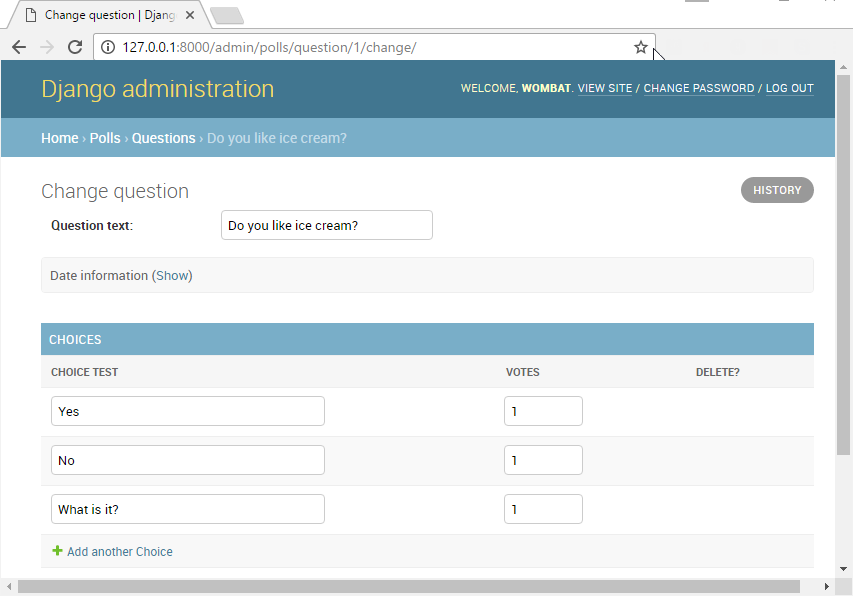
('Date information', {'fields': ['pub\_date'], 'classes': ['collapse']}),

]

inlines = [ChoiceInline]

admin.site.register(Question, QuestionAdmin)

Now look at the Change question page:



Writing views

Open the file polls/views.py for editing and type the following Python code:

from django.http import HttpResponse

def index(request):

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

Next, add a new file to the polls directory with the name urls.py and type the following code in it:

from django.conf.urls import url

from . import views

urlpatterns = [

url(r'^$', views.index, name='index'),

]

Next, open for editing the file mysite/urls.py (which PyCharm has already created for you) and add a URL for the index page. You should end up with the following code:

from django.conf.urls import include, url

from django.contrib import admin

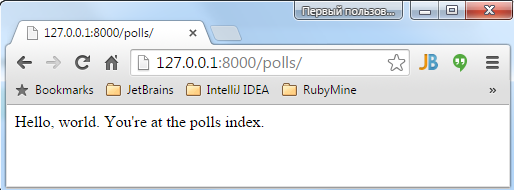
urlpatterns = [

url(r'^polls/', include('polls.urls')), #this line added

url(r'^admin/', admin.site.urls),

]

Now, open the page 127.0.0.1:8000/polls/ and enjoy:



Next, let’s add more views. Again, add the following code to the file polls/views.py:

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 url() calls :

from django.conf.urls import url

from . import views

urlpatterns = [

url(r'^$', views.index, name='index'),

# ex: /polls/5/

url(r'^(?P<question\_id>[0-9]+)/$', views.detail, name='detail'),

# ex: /polls/5/results/

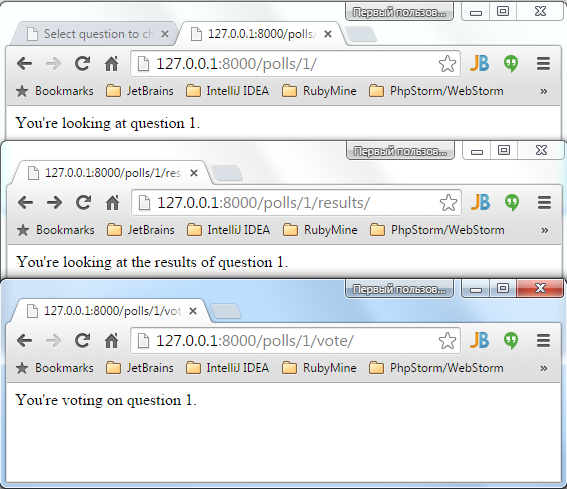
url(r'^(?P<question\_id>[0-9]+)/results/$', views.results, name='results'),

# ex: /polls/5/vote/

url(r'^(?P<question\_id>[0-9]+)/vote/$', views.vote, name='vote'),

]

If you now open the corresponding pages in your browser, you will see, for example:



Creating Django templates

As you see, the design of these pages is hardcoded in views. So, to make it more readable, you have to edit the corresponding Python code. Let’s then separate the visual representation of the output from Python - to do that, let’s create templates.

Open for editing the file polls/views.py and replace its contents with the following code:

from django.http import HttpResponse, HttpResponseRedirect

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

from django.urls import reverse

from .models import Question, Choice

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)

def detail(request, question\_id):

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

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

def results(request, question\_id):

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

return render(request, 'polls/results.html', {'question': 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):

return render(request, 'polls/detail.html', {

'question': question,

'error\_message': "You didn't select a choice.",

})

else:

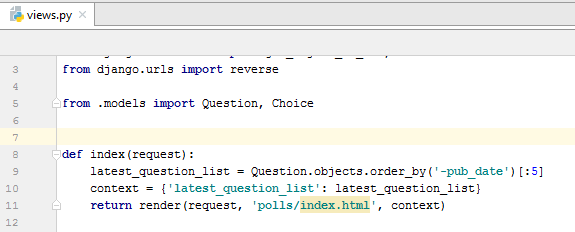
selected\_choice.votes += 1

selected\_choice.save()

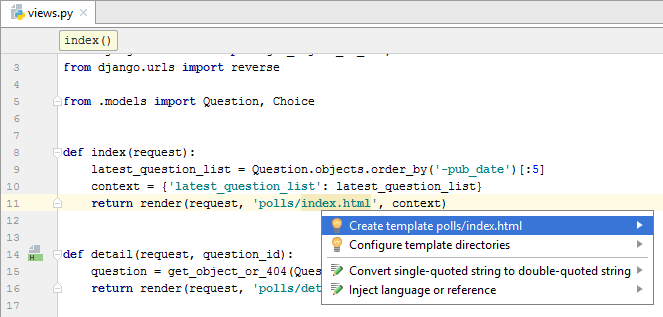
return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))

By the way, pay attention to the import assistant that helps you create import statements.

The first thing you notice is an unresolved reference to the page index.html:



PyCharm suggests a [quick fix](https://www.jetbrains.com/help/pycharm/intention-actions.html): if you click the light bulb, or press Alt+Enter, the corresponding template file is created in the templates folder (note that PyCharm also creates the directory polls where this template should reside):



By now, the file index.html is empty. Add the following code to it:

{% load staticfiles %}

<link rel="stylesheet" type="text/css" href="{% static 'polls/style.css' %}" />

{% if latest\_question\_list %}

<ul>

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

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

{% endfor %}

</ul>

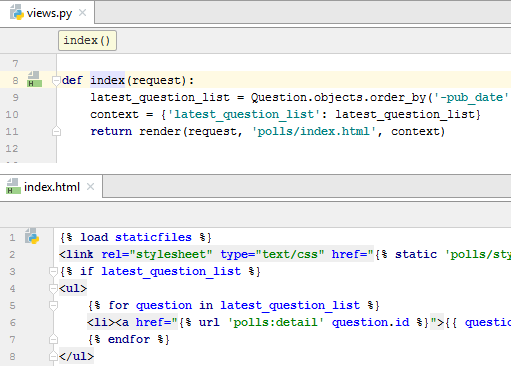
{% else %}

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

{% endif %}

Note [code completion](https://www.jetbrains.com/help/pycharm/auto-completing-code.html) in the template files! For example, when you type the opening {%, PyCharm adds the matching closing one %} automatically, placing the caret at the location of the future input. In HTML tags, code completion is also available.

Pay attention to the icons classTypeHtmland pythonScriptthat appear in the left gutter of the files views.py and index.html respectively:



These icons enable you to jump between a view method and its template straight away. Read more about this kind of navigation in the articles [Navigating Between Templates and Views](https://www.jetbrains.com/help/pycharm/navigating-between-templates-and-views.html) and [Part 6. Django-Specific Navigation](https://www.jetbrains.com/help/pycharm/part-6-django-specific-navigation.html).

Using a Style Sheet

As you can see in the view file index.html, there is a reference to a Style Sheet, and it’s unresolved:

Resolve this reference in the following way:

Create directory. To do that, in the Project view, select the Python package polls, and then press Alt+Insert.

On the pop-up menu that appears, choose Directory, and specify the name of the directory structure static/polls.

Next, create a Style Sheet in this directory. To do that, choose the innermost directory polls, press Alt+Insert, choose the option Stylesheet, and enter style in the dialog box that opens.

Provide some contents to the created Style Sheet, depending on your preferences. For example, we’d like to see a bulleted list of questions colored green:

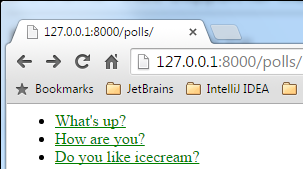
li a {

color: green;

}

Here we are!

Now let's check the list of available polls. Our admin site is already running, and the easiest way to visit the page that contains the list of polls (the index page), is to specify its URL: in the address bar of the browser, instead of /admin/, type /polls/:



Test it…

Now let’s see how PyCharm helps simplify testing your application.

There is already the file tests.py in the polls directory. By now, this file is empty. Naturally, it is advisable to place the new tests in this particular file. For example, we'd like to make sure our poll is not empty:

import datetime

from django.core.urlresolvers import reverse

from django.test import TestCase

from django.utils import timezone

from .models import Question

def create\_question(question\_text, days):

time = timezone.now() + datetime.timedelta(days=days)

return Question.objects.create(question\_text=question\_text, pub\_date=time)

class QuestionViewTests(TestCase):

def test\_index\_view\_with\_no\_questions(self):

"""

If no questions exist, an appropriate message should be displayed.

"""

response = self.client.get(reverse('index'))

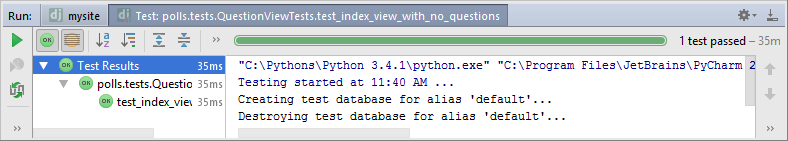
self.assertEqual(response.status\_code, 200)

self.assertContains(response, "No polls are available.")

self.assertQuerysetEqual(response.context['latest\_question\_list'], [])

To run this test, right-click the background of the file tests.py in the editor, choose the option Run, or just press Ctrl+Shift+F10. PyCharm suggests two options: run UnitTest (which is defined as the [default test runner](https://www.jetbrains.com/help/pycharm/python-integrated-tools.html)), or a Django test.

The test results show in the [Test Runner](https://www.jetbrains.com/help/pycharm/test-runner-tab.html) tab of the Run tool window:



Summary

This brief tutorial is over. You have successfully created and launched a simple Django application. Let's repeat what has been done with the help of PyCharm:

A Django project and application have been created

The Django server launched

A database configured

Models, views and templates created

The application launched

Tests created and executed