

# Building A Blog Application With Django

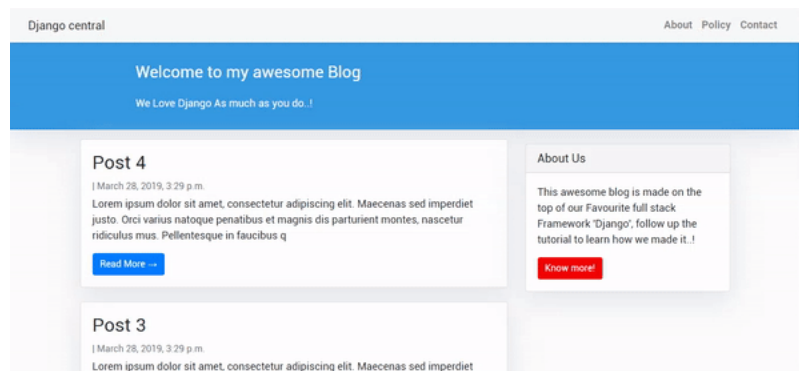


In this tutorial, we'll build a Blog application with Django 2.X that allows users to create, edit, and delete posts. The homepage will list all blog posts, and there will be a dedicated detail page for each individual post. Django is capable of making more advanced stuff but making a blog is an excellent first step to get a good grasp over the framework. The purpose of this chapter is to get a general idea about the working of Django.

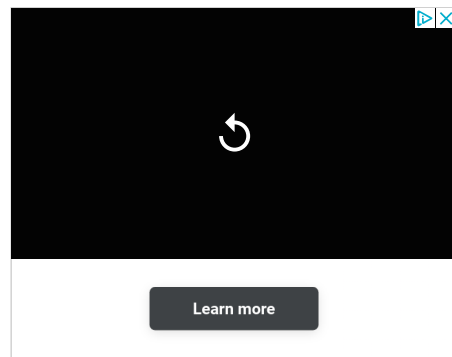
Here is a sneak peek of what we are going to make.

## Sponsors





Before kicking off, I hope you already have a brief idea about the framework we are going to use for this project if not then read the following article: [Django – Web Framework For Perfectionists](#).



## Pre-Requirements

Django is an open-source web framework, written in Python, that follows the model-view-template architectural pattern. So Python is needed to be installed in your machine. Unfortunately, there was a significant update to Python several years ago that created a big split between Python versions namely Python 2 the legacy version and Python 3 the version in active development.

Since Python 3 is the current version in active development and addressed as the future of Python, Django rolled out a significant update, and now all the releases after Django 2.0 are only compatible with Python 3.x. Therefore this tutorial is strictly for Python 3.x. Make sure you have Python 3 installed on your machine if not follow the below guides.

### Windows Users

- [How To Install Python On Windows](#)

### Mac And Unix Users

- [How To Install Python 3 on Mac OS](#)

## Creating And Activating A Virtual Environment

While building python projects, it's a good practice to work in virtual environments to keep your project, and it's dependency isolated on your machine. There is an entire article on the importance of virtual environments Check it out here: [How To A Create Virtual Environment for Python](#)

### Windows Users

```
cd Desktop
virtualenv django
cd django
Scripts\activate.bat
```

### Mac and Unix Users

```
mkdir django
cd django
python3 -m venv myenv
source django/bin/activate
```

Now you should see `(django)` prefixed in your terminal, which indicates that the virtual environment is successfully activated, if not then go through the guide again.

## Installing Django In The Virtual Environment

If you have already installed Django, you can skip this section and jump straight to the Setting up the project section. To Install Django on your virtual environment run the below command

```
pip install Django
```

This will install the latest version of Django in our virtual environment. To know more about Django installation read: [How To Install Django](#)

Note – You must install a version of Django greater than 2.0

## Setting Up The Project

In your workspace create a directory called `mysite` and navigate into it.

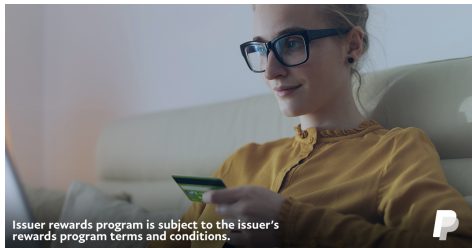
```
cd Desktop
mkdir mysite
cd mysite
```

**Now run the following command in your shell to create a Django project.**

```
django-admin startproject mysite
```

**This will generate a project structure with several directories and python scripts.**

```
├── mysite
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```



Issuer rewards program is subject to the issuer's rewards program terms and conditions.



## Sign Up for PayPal



**Ad** Rack up points, miles and more when you check out with PayPal using a...

PayPal

[Learn more](#)

**To know more about the function of the files read:** [Starting A Django Project](#)

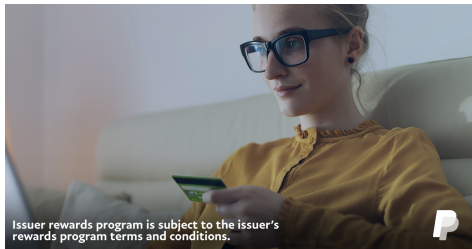
Next, we need to create a Django application called **blog**. A Django application exists to perform a particular task. You need to create specific applications that are responsible for providing your site desired functionalities.

Navigate into the outer directory where `manage.py` script exists and run the below command.

```
cd mysite
python manage.py startapp blog
```

These will create an app named **blog** in our project.

```
|— db.sqlite3
|— mysite
|   |— __init__.py
|   |— settings.py
|   |— urls.py
|   |— wsgi.py
|— manage.py
|— blog
|   |— __init__.py
|   |— admin.py
|   |— apps.py
|   |— migrations
|   |   |— __init__.py
|   |— models.py
|   |— tests.py
|   |— views.py
```



Sign Up for PayPal



**Ad** Rack up points, miles and more when you check out with PayPal using a...

PayPal

Learn more

Now we need to inform Django that a new application has been created, open your `settings.py` file and scroll to the installed apps section, which should have some already installed apps.

```
INSTALLED_APPS = [
    'django.contrib.admin',
```

```
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',

]
```

**Now add the newly created app blog at the bottom and save it.**

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

**Next, make migrations.**

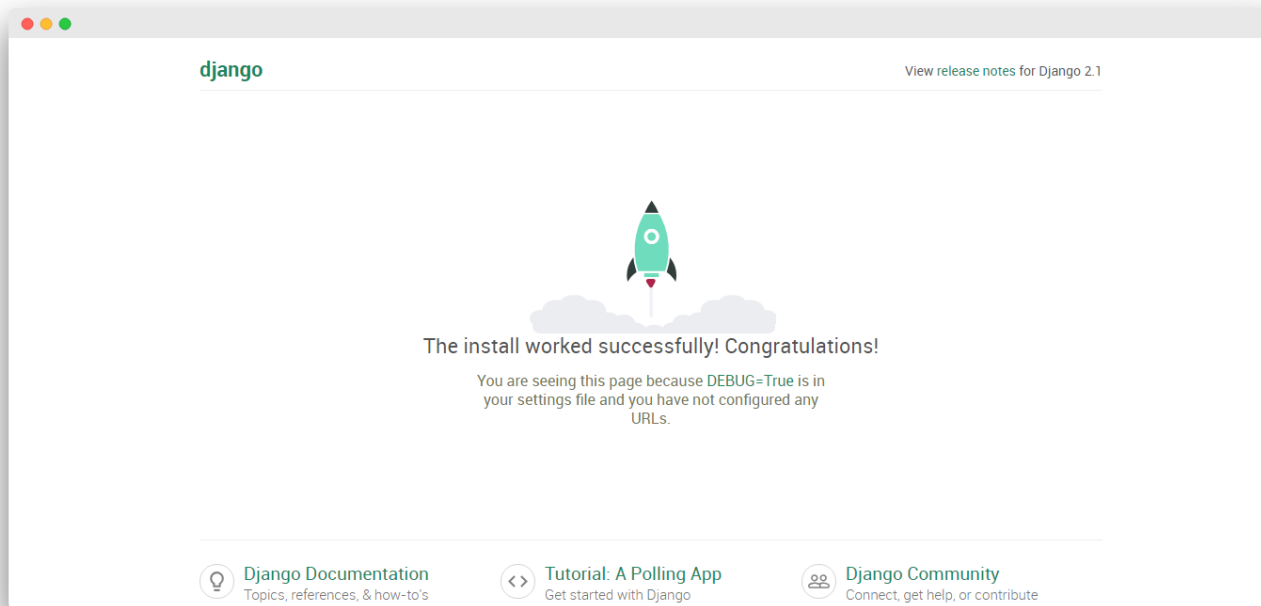
```
python manage.py migrate
```

**This will apply all the unapplied migrations on the SQLite database which comes along with the Django installation.**

**Let's test our configurations by running the Django's built-in development server.**

```
python manage.py runserver
```

**Open your browser and go to this address `http://127.0.0.1:8000/` if everything went well you should see this page.**



## Database Models

Now we will define the data models for our blog. A model is a **Python class** that subclasses `django.db.models.Model`, in which each attribute represents a database field. Using this subclass functionality, we automatically have access to everything within `django.db.models.Models` and can add additional fields and methods as desired. We will have a **Post** model in our database to store posts.

```
from django.db import models
from django.contrib.auth.models import User

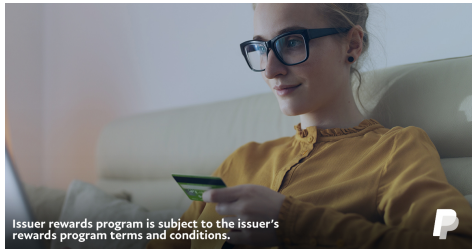
STATUS = (
    (0, "Draft"),
    (1, "Publish")
)

class Post(models.Model):
    title = models.CharField(max_length=200, unique=True)
    slug = models.SlugField(max_length=200, unique=True)
```

```
author = models.ForeignKey(User, on_delete= models.CASCADE,related_name='blog_posts')
updated_on = models.DateTimeField(auto_now= True)
content = models.TextField()
created_on = models.DateTimeField(auto_now_add=True)
status = models.IntegerField(choices=STATUS, default=0)

class Meta:
    ordering = ['-created_on']

def __str__(self):
    return self.title
```



## Sign Up for PayPal

**Ad** Rack up points, miles and more when you check out with PayPal using a...

PayPal

[Learn more](#)

At the top, we're importing the class `models` and then creating a subclass of `models.Model`. Like any typical blog each blog post will have a title, slug, author name, and the timestamp or date when the article was published or last updated.

Notice how we declared a tuple for `STATUS` of a post to keep draft and published posts separated when we render them out with templates.

The `Meta` class inside the model contains metadata. We tell Django to sort results in the `created_on` field in descending order by default when we query the database. We specify descending order using the negative prefix. By doing so, posts published recently will appear first.

The `__str__()` method is the default human-readable representation of the object. Django will use it in many places, such as the administration site.

Now that our new database model is created we need to create a new migration record for it and migrate the change into our database.

```
(django) $ python manage.py makemigrations
(django) $ python manage.py migrate
```

Now we are done with the database.

## Creating An Administration Site



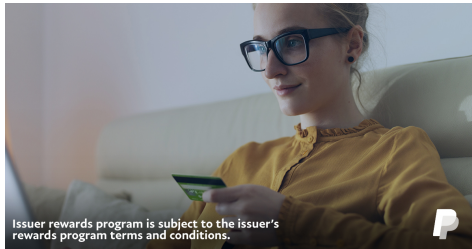
We will create an admin panel to create and manage Posts. Fortunately, Django comes with an inbuilt admin interface for such tasks.

In order to use the Django admin first, we need to create a superuser by running the following command in the prompt.

```
python manage.py createsuperuser
```

You will be prompted to enter email, password, and username. Note that for security concerns Password won't be visible.

```
Username (leave blank to use 'user'): admin
Email address: admin@gamil.com
Password:
Password (again):
```



### Sign Up for PayPal

**Ad** Rack up points, miles and more when you check out with PayPal using a...

PayPal

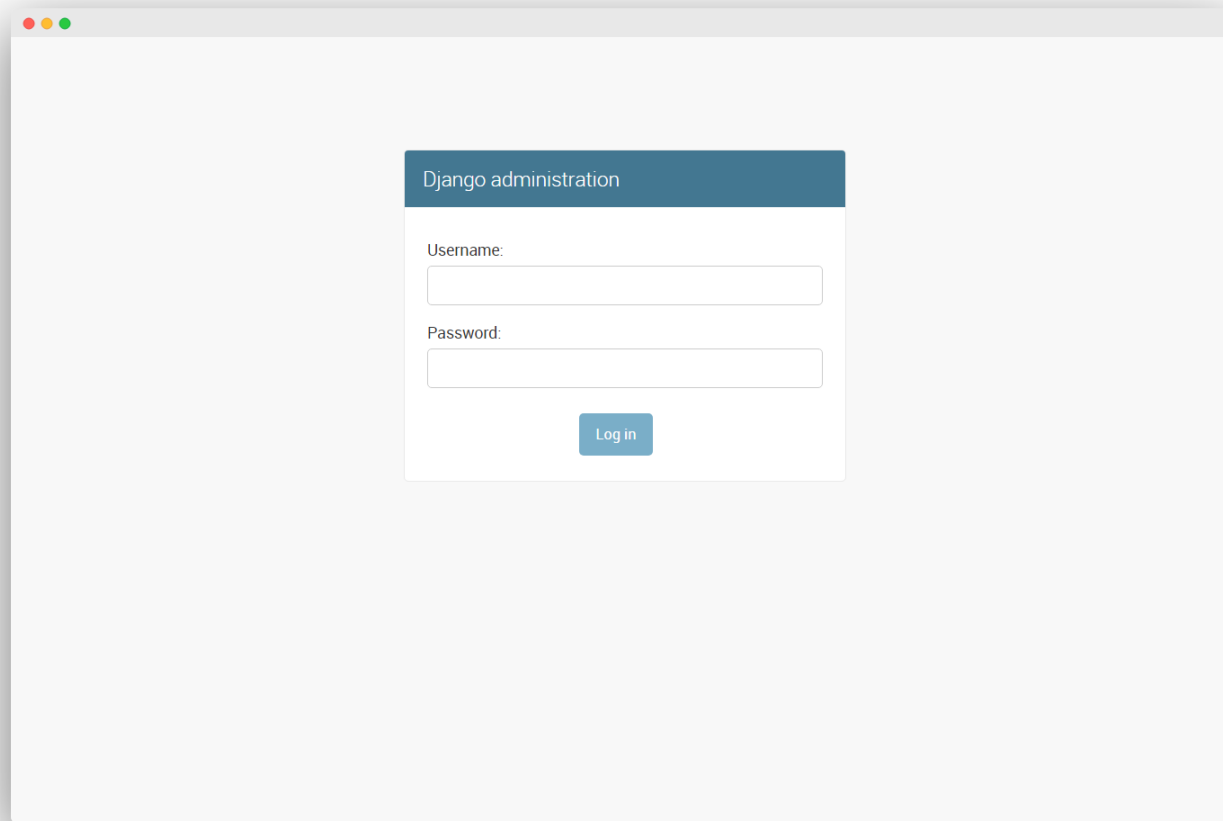
[Learn more](#)

Enter any details you can always change them later. After that rerun the development server and go to the address

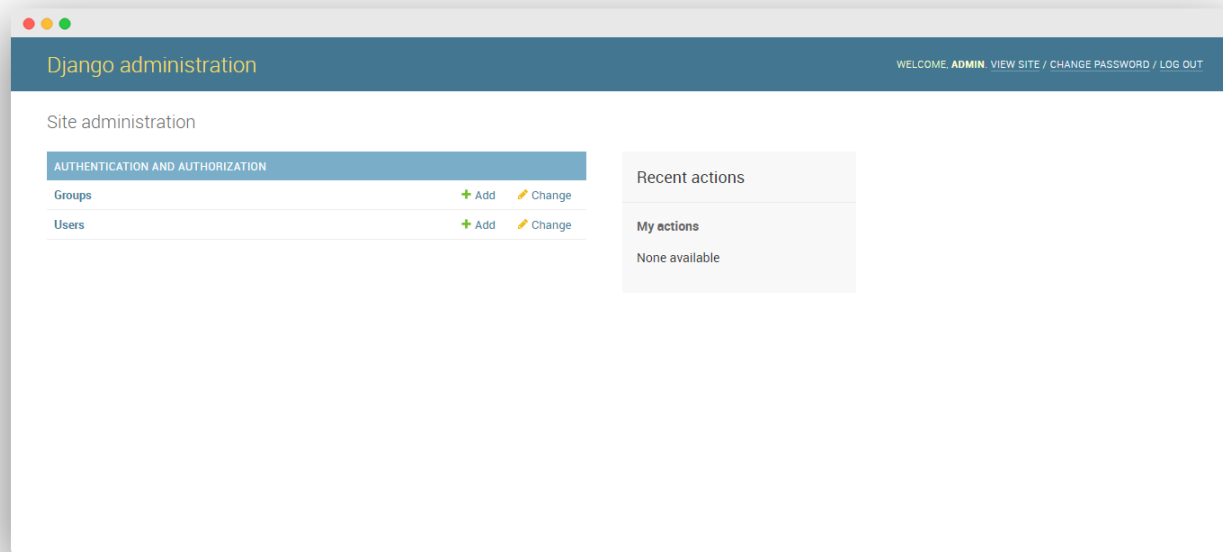
`http://127.0.0.1:8000/admin/`

```
python manage.py runserver
```

You should see a login page, enter the details you provided for the superuser.



**After you log in you should see a basic admin panel with Groups and Users models which come from Django authentication framework located in `django.contrib.auth` .**



Still, we can't create posts from the panel we need to add the Post model to our admin.

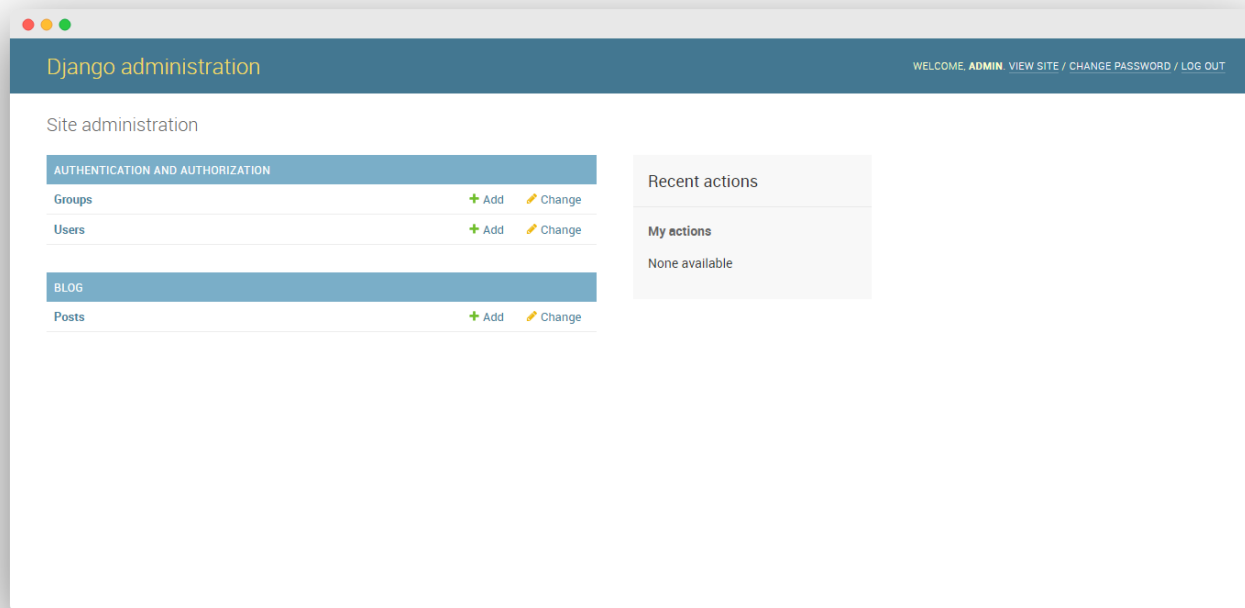
## Adding Models To The Administration Site

Open the `blog/admin.py` file and register the Post model there as follows.

```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

Save the file and refresh the page you should see the Posts model there.



**Now let's create our first blog post click on the Add icon beside Post which will take you to another page where you can create a post. Fill the respective forms and create your first ever post.**

Home » Blog » Posts » Add post

Add post

Title:



My first ever post

Slug:

my-first-ever-post


Author:

admin

Content:

This is my first awesome post..!!



Status:

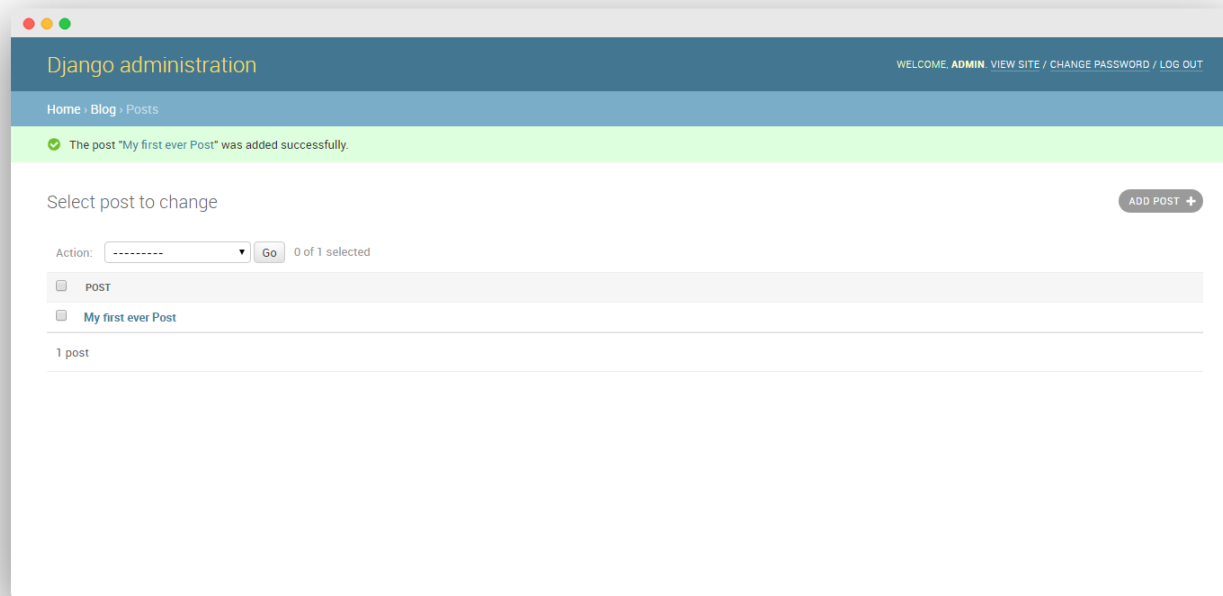
Publish

Save and add another

Save and continue editing

SAVE

Once you are done with the Post save it now, you will be redirected to the post list page with a success message at the top.



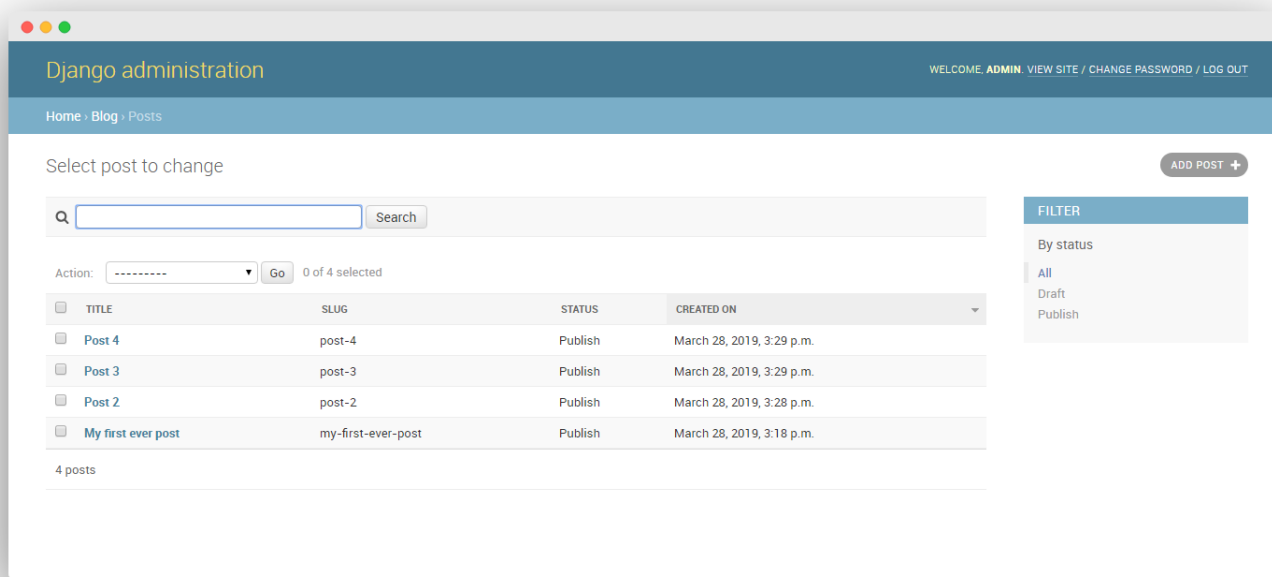
Though it does the work, we can customize the way data is displayed in the administration panel according to our convenience. Open the `admin.py` file again and replace it with the code below.

```
from django.contrib import admin
from .models import Post

class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'slug', 'status', 'created_on')
    list_filter = ("status",)
    search_fields = ['title', 'content']
    prepopulated_fields = {'slug': ('title',)}

admin.site.register(Post, PostAdmin)
```

This will make our admin dashboard more efficient. Now if you visit the post list, you will see more details about the Post.



**Note that I have added a few posts for testing.**

The `list_display` attribute does what its name suggests display the properties mentioned in the tuple in the post list for each post.

If you notice at the right, there is a filter which is filtering the post depending on their Status this is done by the `list_filter` method.

And now we have a search bar at the top of the list, which will search the database from the `search_fields` attributes. The last attribute `prepopulated_fields` populates the slug, now if you create a post the slug will automatically be filled based upon your title.

Now that our database model is complete we need to create the necessary views, URLs, and templates so we can display the information on our web application.

## Building Views

A Django view is just a `Python function` that receives a web request and returns a web response. We're going to use class-based views then map URLs for each view and create an HTML templated for the data returned from the views.

Open the `blog/views.py` file and start coding.

```
from django.views import generic
from .models import Post

class PostList(generic.ListView):
    queryset = Post.objects.filter(status=1).order_by('-created_on')
    template_name = 'index.html'

class PostDetail(generic.DetailView):
    model = Post
    template_name = 'post_detail.html'
```

The built-in ListViews which is a subclass of generic class-based-views render a list with the objects of the specified model we just need to mention the template, similarly DetailView provides a detailed view for a given object of the model at the provided template.

Note that for `PostList` view we have applied a filter so that only the post with status published be shown at the front end of our blog. Also in the same query, we have arranged all the posts by their creation date. The ( - ) sign before the `created_on` signifies the latest post would be at the top and so on.

## Adding URL patterns for Views

We need to map the URL for the views we made above. When a user makes a request for a page on your web app, the Django controller takes over to look for the corresponding view via the `urls.py` file, and then return the HTML response or a 404 not found error, if not found.

Create an `urls.py` file in your blog application directory and add the following code.

```
from . import views
from django.urls import path

urlpatterns = [
    path('', views.PostList.as_view(), name='home'),
    path('<slug:slug>', views.PostDetail.as_view(), name='post_detail'),
]
```

We mapped general URL patterns for our views using the path function. The first pattern takes an empty string denoted by `' '` and returns the result generated from the `PostList` view which is essentially a list of posts for our homepage and at last we have an optional parameter name which is basically a name for the view which will later be used in the templates.



Names are an optional parameter, but it is a good practice to give unique and rememberable names to views which makes our work easy while designing templates and it helps keep things organized as your number of URLs grows.

Next, we have the generalized expression for the `PostDetail` views which resolve the slug (a string consisting of ASCII letters or numbers) Django uses angle brackets `< >` to capture the values from the URL and return the equivalent post detail page.

Now we need to include these blog URLs to the actual project for doing so open the `mysite/urls.py` file.

```
from django.contrib import admin

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

Now first import the include function and then add the path to the new `urls.py` file in the URL patterns list.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('blog.urls')),
]
```

Now all the request will directly be handled by the blog app.

## Creating Templates For The Views

We are done with the Models and Views now we need to make templates to render the result to our users. To use Django templates we need to [configure the template setting](#) first.

Create directory templates in the base directory. Now open the project's `settings.py` file and just below `BASE_DIR` add the route to the template directory as follows.

```
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
```

Now In `settings.py` scroll to the, `TEMPLATES` which should look like this.

```

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        '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',
            ],
        },
    },
]

```

Now add the newly created `TEMPLATE_DIRS` in the `DIRS` .

```

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        # Add 'TEMPLATE_DIRS' here
        'DIRS': [TEMPLATE_DIRS],
        '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',
            ],
        },
    },
]

```

Now save and close the file we are done with the configurations.

Django makes it possible to separate python and HTML, the python goes in views and HTML goes in templates. Django has a powerful template language that allows you to specify how data is displayed. It is based on template tags, template variables, and template

filters.

I'll start off with a `base.html` file and a `index.html` file that inherits from it. Then later when we add templates for homepage and post detail pages, they too can inherit from `base.html` .

Let's start with the `base.html` file which will have common elements for the blog at any page like the navbar and footer. Also, we are using [Bootstrap](#) for the UI and Roboto font.

```
<!DOCTYPE html>
<html>

  <head>
    <title>Django Central</title>
    <link href="https://fonts.googleapis.com/css?family=Roboto:400,700" rel="stylesheet">
    <meta name="google" content="notranslate" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5
      crossorigin="anonymous" />
  </head>

  <body>
    <style>
      body {
        font-family: "Roboto", sans-serif;
        font-size: 17px;
        background-color: #fdfdfd;
      }
      .shadow {
        box-shadow: 0 4px 2px -2px rgba(0, 0, 0, 0.1);
      }
      .btn-danger {
        color: #fff;
        background-color: #f00000;
        border-color: #dc281e;
      }
      .masthead {
        background: #3398E1;
        height: auto;
        padding-bottom: 15px;
        box-shadow: 0 16px 48px #E3E7EB;
        padding-top: 10px;
      }
    </style>
  </body>
</html>
```

```

    }
</style>

<!-- Navigation -->
<nav class="navbar navbar-expand-lg navbar-light bg-light shadow" id="mainNav">
    <div class="container-fluid">
        <a class="navbar-brand" href="{% url 'home' %}">Django central</a>
        <button class="navbar-toggler navbar-toggler-right" type="button" data-toggle="collapse" data-target="#navbarResponsive"
            aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarResponsive">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item text-black">
                    <a class="nav-link text-black font-weight-bold" href="#">About</a>
                </li>
                <li class="nav-item text-black">
                    <a class="nav-link text-black font-weight-bold" href="#">Policy</a>
                </li>
                <li class="nav-item text-black">
                    <a class="nav-link text-black font-weight-bold" href="#">Contact</a>
                </li>
            </ul>
        </div>
    </div>
</nav>

{% block content %}
<!-- Content Goes here -->
{% endblock content %}

<!-- Footer -->
<footer class="py-3 bg-grey">
    <p class="m-0 text-dark text-center ">Copyright &copy; Django Central</p>
</footer>
</body>
</html>

```

This is a regular HTML file except for the tags inside curly braces { } these are called template tags.

The {% url 'home' %} Returns an absolute path reference, it generates a link to the home view which is also the List view for posts.

The `{% block content %}` Defines a block that can be overridden by child templates, this is where the content from the other HTML file will get injected.

Next, we will make a small sidebar widget which will be inherited by all the pages across the site. Notice sidebar is also being injected in the `base.html` file this makes it globally available for pages inheriting the base file.

```
{% block sidebar %}

<style>
    .card{
        box-shadow: 0 16px 48px #E3E7EB;
    }
</style>

<!-- Sidebar Widgets Column -->
<div class="col-md-4 float-right ">
<div class="card my-4">
    <h5 class="card-header">About Us</h5>
    <div class="card-body">
        <p class="card-text"> This awesome blog is made on the top of our Favourite full stack Framework 'Django', follow up the
        <a href="https://djangocentral.com/building-a-blog-application-with-django"
            class="btn btn-danger">Know more!</a>
    </div>
</div>
</div>

{% endblock sidebar %}
```

Next, create the `index.html` file of our blog that's the homepage.

```
{% extends "base.html" %}
{% block content %}
<style>
    body {
        font-family: "Roboto", sans-serif;
        font-size: 18px;
        background-color: #fdfdfd;
    }
</style>
```

```

.head_text {
    color: white;
}

.card {
    box-shadow: 0 16px 48px #E3E7EB;
}
</style>

<header class="masthead">
    <div class="overlay"></div>
    <div class="container">
        <div class="row">
            <div class=" col-md-8 col-md-10 mx-auto">
                <div class="site-heading">
                    <h3 class=" site-heading my-4 mt-3 text-white"> Welcome to my awesome Blog </h3>
                    <p class="text-light">We Love Django As much as you do..! &nbsp;
                </p>
            </div>
        </div>
    </div>
</header>
<div class="container">
    <div class="row">
        <!-- Blog Entries Column -->
        <div class="col-md-8 mt-3 left">
            {% for post in post_list %}
            <div class="card mb-4">
                <div class="card-body">
                    <h2 class="card-title">{{ post.title }}</h2>
                    <p class="card-text text-muted h6">{{ post.author }} | {{ post.created_on}} </p>
                    <p class="card-text">{{post.content|slice:"":200" }}</p>
                    <a href="{% url 'post_detail' post.slug %}" class="btn btn-primary">Read More &rarr;</a>
                </div>
            </div>
            {% endfor %}
        </div>
        {% block sidebar %} {% include 'sidebar.html' %} {% endblock sidebar %}
    </div>
</div>
{%endblock%}

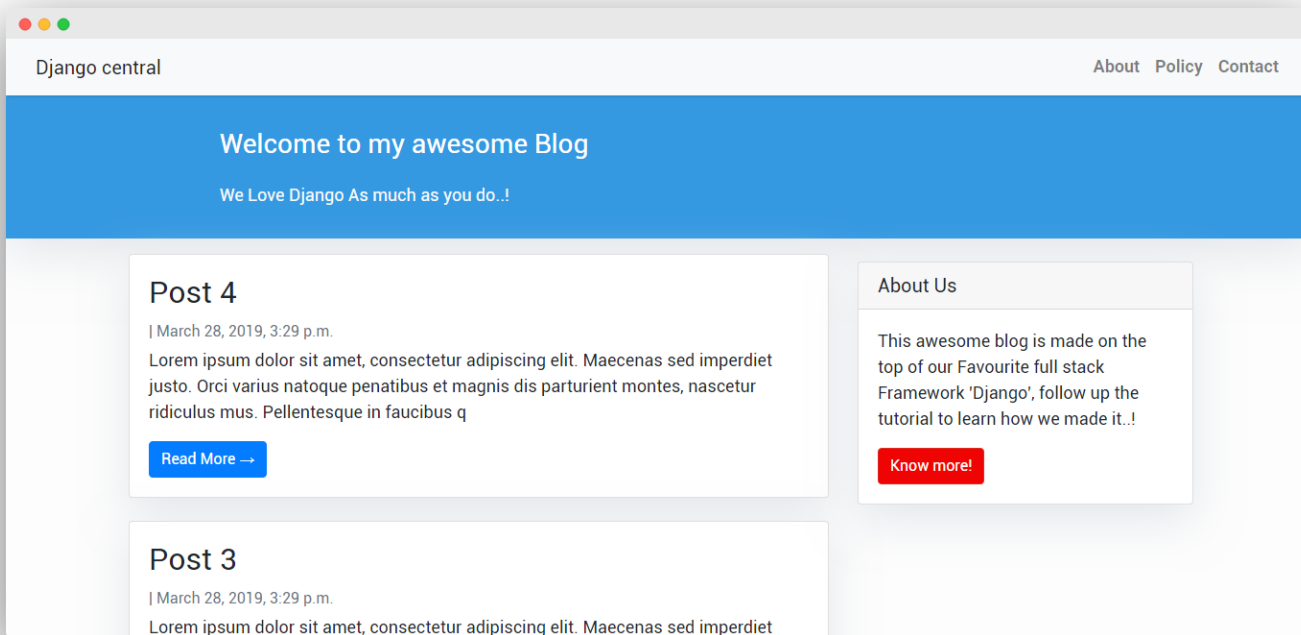
```

With the `{% extends %}` template tag, we tell Django to inherit from the `base.html` template. Then, we are filling the content blocks of the base template with content.

Notice we are using `for loop` in HTML that's the power of Django templates it makes HTML Dynamic. The loop is iterating through the posts and displaying their title, date, author, and body, including a link in the title to the canonical URL of the post.

In the body of the post, we are also using template filters to limit the words on the excerpts to 200 characters. Template filters allow you to modify variables for display and look like `{{ variable | filter }}` .

Now run the server and visit `http://127.0.0.1:8000/` you will see the homepage of our blog.



Looks good..!

You might have noticed I have imported some dummy content to fill the page you can do the same. Now let's make an HTML template for the detailed view of our posts.

Next, Create a file `post_detail.html` and paste the below HTML there.

```
{% extends 'base.html' %} {% block content %}

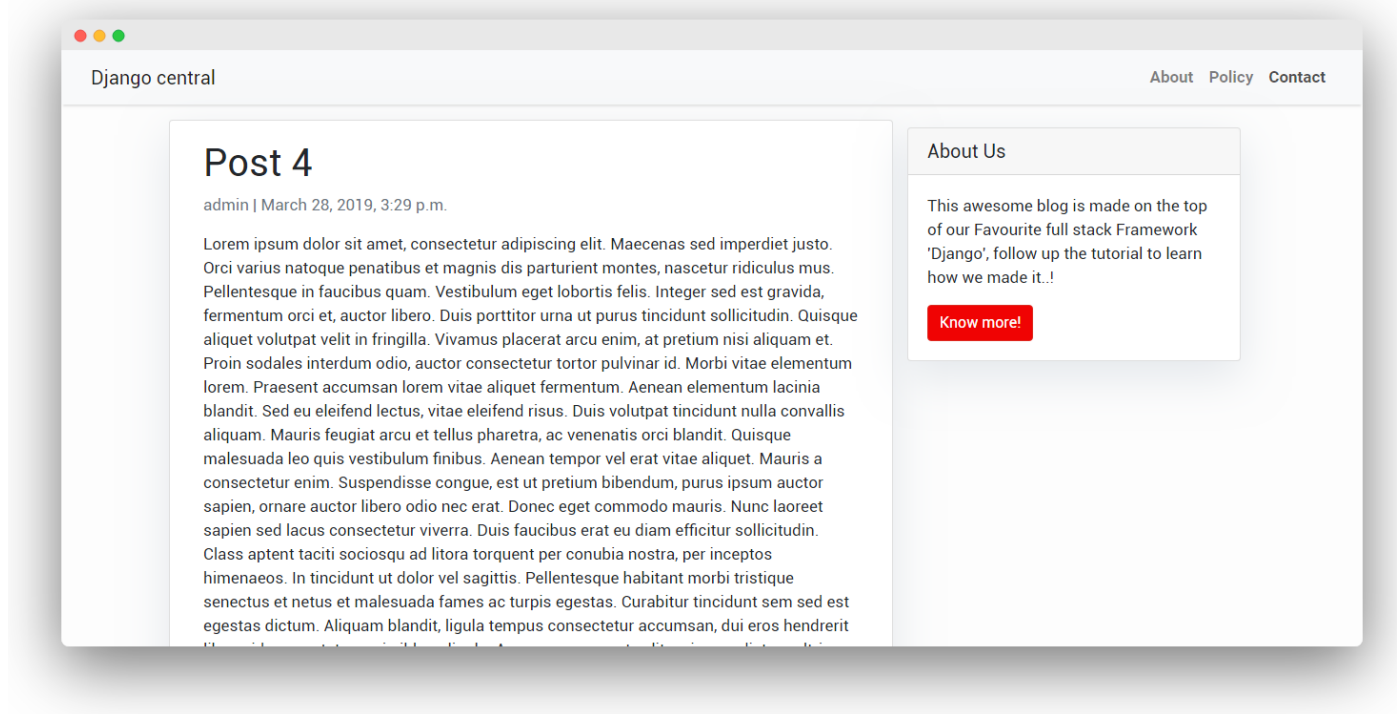
<div class="container">
  <div class="row">
    <div class="col-md-8 card mb-4 mt-3 left top">
      <div class="card-body">
        <h1>{% block title %} {{ object.title }} {% endblock title %}</h1>
        <p class=" text-muted">{{ post.author }} | {{ post.created_on }}</p>
        <p class="card-text ">{{ object.content | safe }}</p>
      </div>
    </div>
    {% block sidebar %} {% include 'sidebar.html' %} {% endblock sidebar %}
  </div>
</div>

{% endblock content %}
```

**At the top, we specify that this template inherits from. `base.html` Then display the `body` from our context object, which `DetailView` makes accessible as an object.**

**Now visit the homepage and click on read more, it should redirect you to the post detail page.**





## Extending The Application

**Adding Comments System** – <https://djangocentral.com/creating-comments-system-with-django/>

**Adding Pagination to the Index page** – <https://djangocentral.com/adding-pagination-with-django/>

**Integrating PostgreSQL with Django** – <https://djangocentral.com/using-postgresql-with-django/>

**Configuring static assets** – <https://djangocentral.com/static-assets-in-django/>

## Wrapping It Up

**We have come to the end of this tutorial. Thank you for reading this far. This post is just the tip of the iceberg considering the number of things you could do with Django.**

**We have built a basic blog application from scratch! Using the Django admin we can create, edit, or delete the content and we used Django's class-based views, and at the end, we made beautiful templates to render it out.**

**If you are stuck at any step, refer to this [GitHub repo](#)**



## Related Articles



Using PostgreSQL with Django



Creating Comments System With Django

## 60 Comments



**Fasho999**

April 1, 2019 at 4:07 am

I was looking for some quality work and here it is! Thanks a lot to Abhijeet for this post 😊

Reply



**Shahidul**

May 11, 2019 at 7:32 pm

Every blog must have comment section but you skip it.

Reply



**Abhijeet**

May 13, 2019 at 9:08 am

I thought it would be too overwhelming for a beginner project but thanks for the feedback.

Reply



Ayo

May 21, 2019 at 2:30 pm

Can you help us with the comment section or a guide we can follow that's easy to understand as yours?

Reply



Mahmoud Gamal

May 25, 2019 at 7:58 am

I agree with you Comment is essential section in every blog

Reply



Michael

August 3, 2019 at 4:21 am

This is a really good post. I really enjoyed going through it.  
Great work!



AdamG

May 12, 2019 at 3:44 am

Thanks for the great start to a blog. How would you go about defining a function for not displaying the "draft" posts from the index page in the views.py? I was trying something like the following but I'm not sure where to enter either the value of 1 or Publish.

```
def get_queryset(self):  
    self.status = get_object_or_404(Status, name=self.kwargs['status'])  
    return Post.objects.filter(status=self.status)
```

As a note in the section where adding the "TEMPLATES\_DIR" the "S" is missing from the word Template in the code example (github has it correctly). Secondly how would you go about defining a function for hiding the "draft" posts from the index page in the view?

Reply



Devil

October 17, 2019 at 5:01 am

We have updated the article.

Reply

Yomi

May 12, 2019 at 11:33 am



Thanks for this info. But where is all the html files created

Reply



Abhijeet

May 13, 2019 at 9:07 am

Template directory holds all the HTML files of the app.

Reply



Ayo

May 18, 2019 at 12:29 am

Hello there, amazing work.

This is my first-time with django and you made it very easy to understand. Thank you.

The only problem i have is my homepage is still showing the default page and not the blog home page, even after i created the templates folder.

Reply



Devil

May 18, 2019 at 2:02 am

Check the repo at Github, you must have missed something.

Reply



Ayo

May 21, 2019 at 2:39 pm

I found the problem, it was from the bootstrap in the html. I was working offline so the bootstrap didn't load.

Thanks alot. Can you help us with creating the comment section?

Reply



Vinay Kumar Mishra

July 20, 2019 at 4:50 pm

I'm having the same problem even when my system is connected to internet, Can you please help me here??

Reply



sang

August 31, 2019 at 4:34 pm

I am working on the comment section once i finished i will post

Reply



AmirAshoori

May 26, 2019 at 8:04 pm

please note that in a fresh django installation, you have to create superuser first then import and use User from auth.models.  
it seems that User is not created before you runb and create admin user (superuser)

Reply



Lucas

May 29, 2019 at 12:30 pm

How do i make a post like this, because on the example is all text, and i'd like to have images, titles, li, and more, as this one.

Reply



Devil

June 12, 2019 at 9:31 am

For that, you have to install a WYSIWYG editor.

Reply



Menja

June 10, 2019 at 7:21 am

Thanks for a good description on how to use build things in Django. I am totally new in this language, but so far I like it!

I do have a problem in the section "Adding Models To The Administration Site"

I have written this in the file:

from django.contrib import admin

admin.site.register(Post)

But I get a NameError: name 'Post' is not defined

Can some one help me with what I am missing?

Reply



Devil

June 12, 2019 at 9:29 am

Run the migrations before adding them into the admin section.

Reply



**Menja**

June 29, 2019 at 11:57 pm

Everything works..

Thanks for the answer and again thanks for a good tutorial!

Do you have any plans for making a tutorial on how to manage images with Django? 😊

Reply



**Devil**

July 12, 2019 at 5:38 am

Yes, I have will publish a tutorial soon.

Reply



**Rambosucks socks**

June 12, 2019 at 9:30 am

by far the best and simple Django illustration. Good job man

Reply



**Devil**

June 13, 2019 at 12:48 pm

I am glad, that you found it helpful.

Reply



**noobjot**

June 18, 2019 at 10:36 am

in the initial registrations of the model in the admins you forgot to import Posts from models  
Anyways great post

Reply



**Devil**

June 23, 2019 at 6:07 am

Thanks for the feedback we have fixed it.

Reply



Vijay

June 28, 2019 at 11:49 am

How can I add feature to add image in between post?

Reply



Vijay

July 1, 2019 at 6:11 am

How we can embed images in blog post ?

Reply



AKASH SENTA

July 1, 2019 at 1:29 pm

Really awesome tutorial and really easy to understand.

Reply



sreekrishnan

July 8, 2019 at 7:13 pm

Awesome and easy to understand tutorial. Thanks for making this page.

Reply



Devil

July 12, 2019 at 5:36 am

You're welcome

Reply



Doug

July 9, 2019 at 12:32 am

I cannot wait to try out this tutorial. I'm coming over from Flask and Django seems to really tie things together well.

Reply



**KATEREGGA FAHAD**

July 10, 2019 at 1:09 pm

hello sir thank u for this tremendous work.....am requesting for your help am stack some where but i dont why.....  
TEMPLATES\_DIR = os.path.join(BASE\_DIR,'templates') where exactly a we supposed to put this code because am pasting it but doesnt work below is where am pasting it but its showing an error when you run the server that invalid syntax help me pliz am new in this MAY  
GOD BLESS YOU WITH MORE KNOWLEDGE  
DATABASES = {  
'default': {  
'ENGINE': 'django.db.backends.sqlite3',  
'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),  
'TEMPLATES\_DIR' = os.path.join(BASE\_DIR,'templates')  
}  
}

Reply



**Devil**

October 15, 2019 at 2:53 pm

In seetings.py file search for TEMPLATES.

Reply



**David**

July 23, 2019 at 12:15 pm

paths doesn't work in django 1.8 so can you do it with urls also? thanks in advanced ☺

Reply



**alpha**

July 29, 2019 at 6:29 am

Hey thanks man for this great tutorial ☺ i am gonna give it try. i will try to extend it. i wanna ask did you develop this your personal blog using django?

Reply



**Devil**

October 15, 2019 at 2:50 pm

No dear, it's powered by WordPress.



Reply



**nishith**

July 30, 2019 at 6:53 am

Thank you for this post. Good starting point

Reply



**Mateo**

July 31, 2019 at 2:43 pm

This is an awesome article and I have lerant a lot but I am not clear on this part: {% for post in post\_list %}  
where did the post\_list variable come from?

Reply



**Devil**

October 15, 2019 at 2:49 pm

From the views

Reply



**Mohan**

August 1, 2019 at 7:51 am

nice documentation

Reply



**Teshuva**

August 31, 2019 at 1:59 pm

perfecto !

Reply



**There Is A Bug**

September 5, 2019 at 10:42 am

Hey, thank you for your wonderful article!

I'm trying to create a blog from your code. However, seems like this code has a problem. The navigation toggle is not working. I search everywhere but still can't fix the problem. Can you take a look at this bug?

Reply



**Gary Wakerley**

September 9, 2019 at 2:41 pm

Thanks for an informative post

Reply



**Shubham**

September 10, 2019 at 6:39 pm

How to allow other people to contribute to the posts as well?  
like allow others to write articles as well and not just super users

Reply



**zee**

September 11, 2019 at 2:00 am

In settings.py  
from  
# Add 'TEMPLATE\_DIR' here  
'DIRS': [TEMPLATE\_DIR],  
make it  
# Add 'TEMPLATES\_DIR' here  
'DIRS': [TEMPLATES\_DIR], <==  
so variable stays consistent throughout.

Reply



**Hemant Gupta**

September 14, 2019 at 9:05 am

Instead of `{% block sidebar %}{% include 'sidebar.html' %}{% endblock sidebar %}` you should use `{% include 'sidebar.html' %}`

Reply

**Vincent**

September 19, 2019 at 10:34 am



Great work. Easy to get along.  
Could you please guide on adding a comment section for a post.

Reply



Artem

October 2, 2019 at 12:32 pm

If you teach people how to create a blog in Django, but your site is create on WP. I do not understand...

Reply



Devil

October 15, 2019 at 2:57 pm

WordPress is open-source, updated regularly and it saves time. If I create my own CMS, I will have to deal with vulnerabilities, script and compatibility issues with browsers and a full-fledged blog like this requires a lot of plugin for optimum performance which would take a lot of time to develop so as I was already familiar with WordPress so instead of reinventing the wheel I choose this as my blogging platform.

Reply



Ruslan

October 3, 2019 at 5:12 pm

Hi there. Thanks for such a detailed guide.

I a have a problem with the last step: it shows empty site without the posts I made in the admin panel.

I guess it might be because I use PostgreSQL and not standard Django DB.  
Migrations are ok, I can see all my articles in the database.

But how do I make them appear on the site?  
Thanks.

Reply



SJ

October 10, 2019 at 11:22 am

Is there any limit to the posts that we can post like 10 , 20 etc ?Thanks beforehand Abhijeet.

Reply



Devil

October 12, 2019 at 12:10 pm

No there are no limits.

Reply



Blake

October 30, 2019 at 5:23 pm

In the database models section, what python file are we editing? I searched around and couldn't find it referenced. Is this still in settings.py?

Reply



Blake

October 30, 2019 at 5:50 pm

I went the the GitHub code and downloaded it. If anyone was wondering on the answer to this question, it is in the models.py file.

Reply



Agbor Chrles

November 19, 2019 at 5:11 am

Hi man sorry for disturbing but i'm getting this error when i run my page after usning the base and the index.html can you help with this error, i am new to django Please

ImproperlyConfigured at /

PostList is missing a QuerySet. Define PostList.model, PostList.queryset, or override PostList.get\_queryset().

Request Method: GET

Request URL: http://127.0.0.1:8000/

Django Version: 2.2.6

Exception Type: ImproperlyConfigured

Exception Value:

PostList is missing a QuerySet. Define PostList.model, PostList.queryset, or override PostList.get\_queryset().

Exception Location: C:\Users\Charles\AppData\Local\Programs\Python\Python37-32\lib\site-packages\django\views\generic\list.py in get\_queryset, line 39

Python Executable: C:\Users\Charles\AppData\Local\Programs\Python\Python37-32\python.exe

Python Version: 3.7.1

Python Path:

```
['D:\\Django Project\\MySite',  
'C:\\Users\\Charles\\AppData\\Local\\Programs\\Python\\Python37-32\\python37.zip',  
'C:\\Users\\Charles\\AppData\\Local\\Programs\\Python\\Python37-32\\DLLs',  
'C:\\Users\\Charles\\AppData\\Local\\Programs\\Python\\Python37-32\\lib',  
'C:\\Users\\Charles\\AppData\\Local\\Programs\\Python\\Python37-32',  
'C:\\Users\\Charles\\AppData\\Local\\Programs\\Python\\Python37-32\\lib\\site-packages',  
'C:\\Users\\Charles\\AppData\\Local\\Programs\\Python\\Python37-32\\lib\\site-packages\\win32',  
'C:\\Users\\Charles\\AppData\\Local\\Programs\\Python\\Python37-32\\lib\\site-packages\\win32\\lib',  
'C:\\Users\\Charles\\AppData\\Local\\Programs\\Python\\Python37-32\\lib\\site-packages\\Pythonwin']
```

Server time: Tue, 19 Nov 2019 05:05:27 +0000

Reply



Abhijeet

November 20, 2019 at 2:10 am

Try matching your code with the Github repo.

Reply



Peebee

November 28, 2019 at 12:40 am

Hi, thanks for the explicit tutorial. However I'd like to see how pictures are incorporated with the models so that a blog post can have a picture also display and not text only. Thanks

Reply



sjr

November 30, 2019 at 2:44 pm

Hey

superb tutorial is easy to follow compared to the documentation, i have a problem with the urls

SystemCheckError: System check identified some issues:

ERRORS:

?: (urls.E004) Your URL pattern [ , <URLPattern '/' [name='post\_detail']>] is invalid. Ensure that urlpatterns is a list of path() and/or re\_path() instances.

i just copy pasted the url file from the blog so can you please help me

Reply



sjr

November 30, 2019 at 3:12 pm

for anyone getting the same problem check your brackets in the urls.py file of your blogs.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked \*

Name \*

Email \*

Website

Post Comment

About

Django Central is an educational site providing content on Python programming and web development to all the programmers and budding programmers across the internet. The main goal of this site is to provide quality tips, tricks, hacks, and other Programming resources that allows beginners to improve their skills.

Categories

- Django
- Programs
- Python
- Tools

Site Links

- About Us
- Contact Us
- Disclaimer
- Privacy Policy