How to add Pagination in Django Project?

Last Updated : 07 Aug, 2023

Pagination system is one of the most common features in blogs, search engine , list of result etc. Seeing the popularity of pagination system django developers have build a Paginator class so that web developers do not have to

Paginator Class live in django/core/paginator.py . So to use Paginator Class we first need to import it from **django.core.paginator**

```
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
```

Syntax :

```
p = Paginator(list_of_objects, no_of_objects_per_page)
```

The first argument is the list of objects which will be distributed over pages. The second argument denotes the number of objects that will be displayed on each page. These two arguments are required.

However Paginator Class takes two more optional arguments which are listed below –

- **orphans** – its value must be an integer less than the no_of_objects_per_page value. It tells if the last page has minimum number of objects. If the number of remaining objects in the last page is less than or equal to the value of this argument then those objects will be added to the previous page. Default value is 0.
- **allow_empty_first_page** – It takes Boolean values. Whether or not the first page is allowed to be empty.

Note : the first argument need not to be a list. Instead it can be a tuple, queryset or other sliceable object with a count() or __len__() method.

How to use Paginator Class?

Suppose we are developing a blogging website. We have defined Post model in models.py and we have creates 8 such posts. Now in views.py, we have written the following code –

Python3

```
from django.shortcuts import render
from .models import Post
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
# Create your views here.


def index(request):
    posts = Post.objects.all()  # fetching all post objects from database
    p = Paginator(posts, 5)  # creating a paginator object
    # getting the desired page number from url
    page_number = request.GET.get('page')
    try:
        page_obj = p.get_page(page_number)  # returns the desired page object
    except PageNotAnInteger:
        # if page_number is not an integer then assign the first page
        page_obj = p.page(1)
    except EmptyPage:
        # if page is empty then return last page
        page_obj = p.page(p.num_pages)
    context = {'page_obj': page_obj}
    # sending the page object to index.html
    return render(request, 'index.html', context)

  #edited by B C SAMRUDH
```

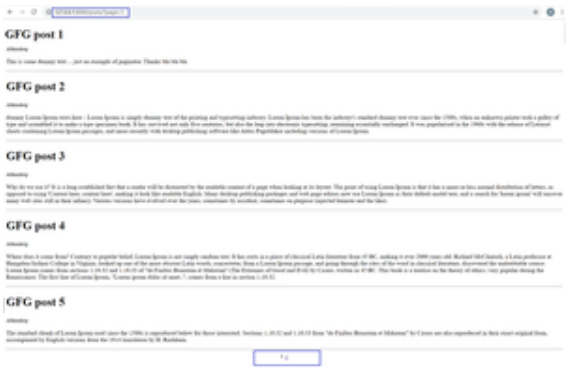
At the third line , Paginator Class is imported. In the index function we have constructed a paginator object called p . This paginator creates page objects. Each Page object will have equal number of post objects. Then we retrieved the desired page number from the query parameter ‘page’ from a GET request. This page number is used to extract the correct page object.

Now in index.html –

HTML

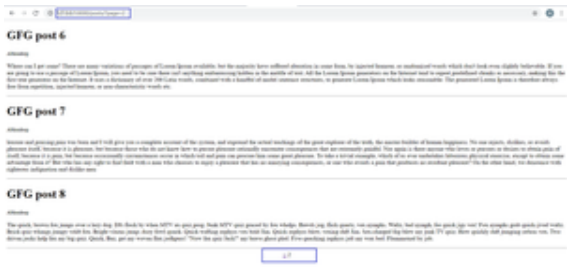
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Django Paginator example</title>
</head>
<body>
    <div class="container">
        {% for post in page_obj.object_list %}
          {# note that the list of posts are in the page_obj.object_list not page_
            <h1>{{post.title}}</h1>
            <small>{{post.author}}</small>

<p>{{post.content}}</p>

            <hr/>
         {% endfor %}
    </div>
    <center>
        {%if page_obj.has_previous %} {# whether the previous page exists #}
            <a href="?page={{page_obj.previous_page_number}}"><<</a> {# link to the
        {% endif %}
        <span>{{page_obj.number}}</span> {# the current page number #}

        {%if page_obj.has_next %} {# whether the next page exists #}
            <a href="?page={{page_obj.next_page_number}}">>></a> {# link to the nex
        {% endif %}
    </center>
</body>
</html>
```

Result :



In the image we have send the value of the page number with a GET request (denoted with rectangle). You can see the pagination in the bottom of the image (marked with rectangle).

Here is another image of the last page –



Methods :

Paginator.get_page(page_number)	It takes a number argument and returns a Page object with the given 1-based index. If the passed argument is not a number then it returns the first page. If the page number is negative or a number that is greater than the number of pages then it returns the last page. Throws EmptyPage error if the object list of the page is empty and allow_empty_first_page is set to false in Paginator object.
Paginator.page(number)	It also does the same thing but raises InvalidPage error if the page of that number does not exist.

Attributes :

Paginator.count	returns the total number of objects across all pages.
Paginator.num_pages	returns the total number of pages
Paginator.page_range	returns a 1-based range iterator

However, as Paginator class uses Page class to distribute objects, It will be better if we know more about the Page Class.

Page Class :

Generally Page object is used in Paginator Class. You rarely have to construct it manually.

Syntax :

```
page = Page( object_list , number, paginator)
```

Here object list is the list of objects , number argument is used to number the Page object. Paginator is the Paginator objects for whom this page object is constructed.

Attributes :

Page.object_list	returns the list of objects
Page.number	returns the number of the page object
Page.paginator	returns the corresponding paginator object

Methods :

Page.has_next()	returns True if the next Page object exits else returns False
Page.has_previous()	returns True if the previous Page object exits else returns False
Page.has_other_pages()	Returns True if there's a next or previous page.
Page.next_page_number()	Returns the next page number. Raises InvalidPage if next page doesn't exist.

Page.previous_page_number()	Returns the previous page number. Raises InvalidPage if previous page doesn't exist.
Page.start_index()	Returns the 1-based index of the first object on the page, relative to all of the objects in the paginator's lis
Page.end_index()	Returns the 1-based index of the last object on the page, relative to all of the objects in the paginator's list.

Are you ready to elevate your web development skills from foundational knowledge to advanced expertise? Explore our Mastering Django Framework - Beginner to Advanced Course on GeeksforGeeks, designed for aspiring developers and experienced programmers. This comprehensive course covers everything you need to know about Django, from the basics to advanced features. Gain practical experience through **hands-on projects** and real-world applications, mastering essential Django principles and techniques. Whether you're just starting or looking to refine your skills, this course will empower you to build sophisticated web applications efficiently. Ready to enhance your web development journey? Enroll now and unlock your potential with Django!

A riter79 6

Previous Article

How to get JSON data from request in Django?

Next Article

Similar Reads

Adding Tags Using Django-Taggit in Django Project

Django-Taggit is a Django application which is used to add tags to blogs, articles etc. It makes very easy for us to make adding the tags functionality to our djang...

2 min read

Adding Pagination in APIs - Django REST Framework

Imagine you have huge amount of details in your database. Do you think that it is wise to retrieve all at once while making an HTTP GET request? Here comes the...

8 min read

Use Pagination with Django Class-Based Generic ListView

Pagination is a crucial feature for any web application that displays a large set of data. It helps manage the data load by breaking it into smaller, more manageabl...

3 min read

How to add RSS Feed and Sitemap to Django project?

This article is in continuation of Blog CMS Project in Django. Check this out here - Building Blog CMS (Content Management System) with Django RSS (Really...

3 min read

How to add Site Header, Site Title, Index Title in a Django Project?

Automatic admin interface one of the most powerful parts of Django. Metadata is read from your models to provide a quick, model-centric interface where trusted...

1 min read

How To Add Unit Testing to Django Project

Unit testing is the practice of testing individual components or units of code in isolation to ensure they function correctly. In the context of Django, units typicall...

4 min read

How to add AMP to Django Project?

A blog mostly needs content but that doesn't mean, your blog will be on top of Google search. For this you will need Speed, Security, user base and first of all...

4 min read

How to customize Django forms using Django Widget Tweaks ?

Django forms are a great feature to create usable forms with just few lines of code. But Django doesn't easily let us edit the form for good designs. Here, we...

3 min read

Styling Django Forms with django-crispy-forms

Django by default doesn't provide any Django form styling method due to which it takes a lot of effort and precious time to beautifully style a form. django-crispy...

1 min read

Integrating Django with Reactjs using Django REST Framework

In this article, we will learn the process of communicating between the Django Backend and React js frontend using the Django REST Framework. For the sake...

15+ min read

Article Tags : Python Technical Scripter Python Django Technical Scripter 2020

Practice Tags : python