Add the slug field inside Django Model

Last Updated : 11 Mar, 2024

The slug field within Django models is a pivotal step for improving the structure and readability of URLs in web applications. This addition allows developers to automatically generate URL-friendly slugs based on titles, enhancing user experience and search engine optimization (SEO). By implementing this feature, you can create cleaner, more meaningful, and SEO-friendly URLs for your content, which is essential for attracting and retaining website visitors.

Slug field in Django

Let's assume our blog has a post with the title 'The Django book by Geeksforgeeks' with primary key id= 2. We might refer to this post with

```
www.geeksforgeeks.org/posts/2.
```

Or, we can reference the title like

```
www.geeksforgeeks.org/posts/The Django book by Geeksforgeeks.
```

But the problem is spaces are not valid in URLs, they need to be replaced by *%20* which is ugly, making it the following

```
www.geeksforgeeks.org/posts/The%20Django%20book%20by%20geeksforgeeks
```

But it is not solving meaningful URL. Another option can be

```
www.geeksforgeeks.org/posts/the-django-book-by-geeksforgeeks
```

So, the slug is now *the-django-book-by-geeksforgeeks*. All letters are down cased and spaces are replaced by hyphens –

Adding Slug field in Django Models

The slug field, represented as models.SlugField in Django models, is used to store a URL-friendly version of a text-based field, such as a title. Its primary purpose is to create cleaner, more readable, and search engine-friendly URLs for your content.

Python3

```
STATUS_CHOICES = (
('draft', 'Draft'),
('published', 'Published'),
)

class Post(models.Model):
title = models.CharField(max_length = 250)
slug = models.SlugField(max_length = 250, null = True, blank = True)
text = models.TextField()
published_at = models.DateTimeField(auto_now_add = True)
updated = models.DateTimeField(auto_now = True)

status = models.CharField(max_length = 10, choices = STATUS_CHOICES,
                                                    default ='draft')


class Meta:
    ordering = ('-published_at', )

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

Adding Slugify to our Project

Now we need to find a way to convert the title into a slug automatically. We want this script to be triggered every time a new instance of *Post* model is created. For this purpose, we will use signals.

Note: Add new file util.py in the same directory where settings.py file is saved.

Python3

```
import string, random
from django.db.models.signals import pre_save
from django.dispatch import receiver
from django.utils.text import slugify

def random_string_generator(size = 10, chars = string.ascii_lowercase + string.di
    return ''.join(random.choice(chars) for _ in range(size))

def unique_slug_generator(instance, new_slug = None):
    if new_slug is not None:
        slug = new_slug
    else:
        slug = slugify(instance.title)
    Klass = instance.__class__
    max_length = Klass._meta.get_field('slug').max_length
    slug = slug[:max_length]
    qs_exists = Klass.objects.filter(slug = slug).exists()

    if qs_exists:
        new_slug = "{slug}-{randstr}".format(
            slug = slug[:max_length-5], randstr = random_string_generator(size =

        return unique_slug_generator(instance, new_slug = new_slug)
    return slug
```

Signals in Django

In many cases when there is a modification in a model's instance we need to execute some action. Django provides us with an elegant way to handle these situations. The signals are utilities that allow associating events with actions. We can develop a function that will run when a signal calls it.

In models.py file of posts app where Post Model was defined, add this in the same file:

Python3

```
@receiver(pre_save, sender=Post)
def pre_save_receiver(sender, instance, *args, **kwargs):
   if not instance.slug:
       instance.slug = unique_slug_generator(instance)
```

The *pre_save_receiver* function should be placed separately outside the Post model.

Modify URL with Slug

To modify your urls.py file to use the slug field in your Django model for generating URLs, you can create URL patterns that include the slug as a parameter. Here's an example of how to do this:

Python3

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

urlpatterns = [
    path('posts/<slug:slug>/', views.post_detail, name='post_detail'),
    # Other URL patterns
]
```

Modify Views

This `detail` view function in Django takes a `slug` parameter from the URL and searches for a post with a matching slug in a case-insensitive manner. If a post is found, it retrieves and renders the post's details using the 'details.html' template. If no matching post is found, it returns an "Post Not Found" response to inform users of the absence of the requested content.

Note: In urls.py edit detail path with path('posts/', detail). In views.py edit the detail function with

Python3

```
def detail(request, slug):
    # Filter posts based on the slug (case-insensitive)
    q = Post.objects.filter(slug__iexact=slug)

    if q.exists():
        # If a post with the given slug exists, retrieve the first matching post
        q = q.first()
    else:
        # If no post is found, return an "Post Not Found" response
        return HttpResponse('<h1>Post Not Found</h1>')

    # Create a context dictionary containing the retrieved post
    context = {'post': q}

    # Render the 'details.html' template with the context
    return render(request, 'posts/details.html', context)
```

The last step is to add the link in HTML file <a href="/posts/{{ a.slug }}" class="btn btn-primary">View</a>. Now we are ready to go to 127.0.0.1:8000/posts/title-you-have-added and it will show you the page details.html.

I ishw...

Previous Article

Django Basic App Model - Makemigrations and Migrate

Next Article

Intermediate fields in Django | Python

Similar Reads

Set a Default Value for a Field in a Django Model

Django is a powerful web framework for building web applications in Python. One common task in Django is setting up models, which are classes that...

4 min read

Limit the Maximum Value of a Numeric Field in a Django Model

Limiting the maximum value of a numeric field in a Django model is essential for maintaining data integrity and preventing errors. By using Django's built-in...

3 min read

How to Make the Foreign Key Field Optional in Django Model?

When working with the relational database, we often need to add foreign key relationships between tables. However, when working with Django, we can...

4 min read

CRUD Operation On A Django Model With A Many-to-Many Field

Django provides a powerful and flexible way to work with relational databases through its Object-Relational Mapping (ORM). One of the common relationships...

4 min read

Understanding Django: Model() vs Model.objects.create()

In Django, when working with models, understanding the differences between Model() and Model.objects.create() is crucial for effective database operations....

3 min read

Django Admin - Disable the 'Add' Action for a Specific Model

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It's designed to help developers take...

3 min read

Built-in Field Validations - Django Models

Built-in Field Validations in Django models are the default validations that come predefined to all Django fields. Every field comes in with built-in validations fro...

3 min read

How to use Django Field Choices ?

Django Field Choices. According to documentation Field Choices are a sequence consisting itself of iterables of exactly two items (e.g. [(A, B), (A, B) ...]) to use as...

2 min read

default - Django Built-in Field Validation

Built-in Field Validations in Django models are the validations that come predefined to all Django fields. Every field comes in with built-in validations fro...

3 min read

blank=True - Django Built-in Field Validation

Built-in Field Validations in Django models are the default validations that come predefined to all Django fields. Every field comes in with built-in validations fro...

4 min read

Article Tags :	Python
Practice Tags :	python