MYSORE UNIVERSITY SCHOOL OF ENGINEERING

Manasagangotri campus, Mysuru-570006 (Approved by AICTE, New Delhi)

UNIVERSITY OF MYSORE

Full Stack Development(21CD71) Assessment Report On:

"Multi-Page Blogging System"

Under the guidance :
Mr. Karthik M N
Assistant Professor,
Department of Computer Science & Design,
MUSE.

Submitted by:
NISHCHITHA GOWDA S
Reg No : 21SECD27

Q.5 Create a Multi-Page Blogging System with the following features:

Users can write blog posts containing a title, author, content, and published date.

 Use Django's generic CreateView, ListView, and DetailView to manage blogs.

 Implement multiple URL configurations:

 /blogs/ → List all blog posts

 /blogs/<int:id>/ → Display a single blog post

 /blogs/new/ → Allow users to create a new blog post

 Use reverse_lazy() to redirect users to the blog list after successfully posting an article.

Introduction:

This Django-based Blogging System enables users to create and manage blog posts. It supports the following features:

- Users can write blog posts containing a title, author, content, and published date.
- The system uses Django's generic views (CreateView, ListView, and DetailView) to manage blogs.
- Multiple URL configurations are implemented:
    - /blogs/ – List all blog posts.
    - /blogs/<int:pk>/ – Display a single blog post.
    - /blogs/new/ – Allow users to create a new blog post.
- After submission, users are redirected to the blog list page using reverse_lazy() with a success message.

Project Overview:

```
blog_system/
├── manage.py
├── db.sqlite3
├── blog_system/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── asgi.py
│   ├── wsgi.py
├── blog/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   ├── views.py
│   ├── urls.py
│   ├── migrations/
│   └── templates/
│       └── blog/
│           ├── blog_list.html
│           ├── blog_detail.html
│           └── blog_form.html
├── ven
v/ (Virtual Environment)
```

Detailed Steps of Implementation:

Step 1: Install Django and Create a Virtual Environment

- Create a virtual environment:

python -m venv venv

Activate the virtual environment:

- **On Windows:**

venv\Scripts\activate

- **On macOS/Linux:**

source venv/bin/activate

Install Django:

pip install Django

Step 2: Create a Django Project

django-admin startproject blog_system

cd blog_system

Step 3: Create a Django App

python manage.py startapp blog

Step 4: Configure settings.py

- Open blog_system/settings.py and add 'blog' to the INSTALLED_APPS list.
- Ensure APP_DIRS is set to True in the TEMPLATES setting.

Step 5: Create the Blog Post Model

- **In blog/models.py, define**

```
# Create your models here.
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    content = models.TextField()
```

```
    published_date = models.DateTimeField() # Enter manually (e.g., "2025-02-23 14:30:00")

    def __str__(self):
        return self.title

```

Run migrations:

python manage.py makemigrations

python manage.py migrate

Step 6: Register the Model in Admin

- **In blog/admin.py, register the model:**

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

class BlogPostAdmin(admin.ModelAdmin):
    list_display = ('title', 'author', 'published_date')  # Display these fields in the list
    search_fields = ('title', 'author')  # Enable search by title or author
    list_filter = ('published_date',)  # Filter posts by date

admin.site.register(BlogPost, BlogPostAdmin)

```

Step 7: Create Views for the Blogging System

- **In blog/views.py, implement:**

```
from django.shortcuts import render
from django.urls import reverse_lazy
from django.views.generic import ListView, DetailView, CreateView
from django.contrib import messages
from .models import BlogPost

class BlogListView(ListView):
    model = BlogPost
    template_name = 'blog/blog_list.html'
    context_object_name = 'blogs'

class BlogDetailView(DetailView):
    model = BlogPost
    template_name = 'blog/blog_detail.html'
```

```
    context_object_name = 'blog'

class BlogCreateView(CreateView):
    model = BlogPost
    template_name = 'blog/blog_forms.html'
    fields = ["title", "author", "content", "published_date"]
    success_url = reverse_lazy('blog_list')

    def form_valid(self, form):
        response = super().form_valid(form)
        messages.success(self.request, "Blog post successfully submitted!")
        return response
```

Step 8: Configure URLs

- **In blog/urls.py:**

```
from django.urls import path
from .views import BlogListView, BlogDetailView, BlogCreateView

urlpatterns = [
    path('', BlogListView.as_view(), name='blog_list'),
    path('<int:pk>/', BlogDetailView.as_view(), name='blog_detail'),
    path('new/', BlogCreateView.as_view(), name='blog_create'),
]
```

- **In blog_system/urls.py:**

```
from django.contrib import admin
from django.urls import path, include
from django.views.generic import RedirectView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('blogs/', include('blog.urls')),  # Blog app URLs
    path('', RedirectView.as_view(url='/blogs/', permanent=True)),  # Redirect '/' to '/blogs/'
]
```

Step 9: Create HTML Templates

- **blog_list.html:**

```
<!DOCTYPE html>
<html>
<head>
<!DOCTYPE html>
<html>
<head>
    <title>Blog List</title>
</head>
<body>
    <!-- Display messages -->
    {% if messages %}
      <ul class="messages">
        {% for message in messages %}
         <li{% if message.tags %} class="{{ message.tags }}"{% endif %}>
            {{ message }}
          </li>
        {% endfor %}
      </ul>
    {% endif %}

    <h1>All Blog Posts</h1>
    <a href="{% url 'blog_create' %}">Create New Blog Post</a>
    <hr>
    <p>Number of blogs: {{ blogs|length }}</p>
    {% if blogs %}
        <ul>
            {% for blog in blogs %}
                <li>
                    <a href="{% url 'blog_detail' blog.pk %}">{{ blog.title }}</a>
                    by {{ blog.author }} on {{ blog.published_date }}
                </li>
            {% endfor %}
        </ul>
    {% else %}
        <p>No blog posts available. Be the first to create one!</p>
    {% endif %}
</body>
</html>
```

- **blog_detail.html:**

<!DOCTYPE html>
<html>

```
<head>
  <title>{{ blog.title }}</title>
</head>
<body>
  <h1>{{ blog.title }}</h1>
  <p><strong>Author:</strong> {{ blog.author }}</p>
  <p><strong>Published Date:</strong> {{ blog.published_date }}</p>
  <p>{{ blog.content }}</p>

  <hr>
  <a href="{% url 'blog_list' %}">Back to Blog List</a>
</body>
</html>
```

- **blog_forms.html:**

```
<!DOCTYPE html>
<html>
<head>
  <title>Create a New Blog Post</title>
</head>
<body>
  <h1>Create a New Blog Post</h1>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
  </form>
  <hr>
  <a href="{% url 'blog_list' %}">Back to Blog List</a>
</body>
</html>
```

Step 10: Create a Superuser (Optional for Admin Access)

- Run:

```
python manage.py createsuperuser
```

Step 11: Run the Django Development Server

- Run:

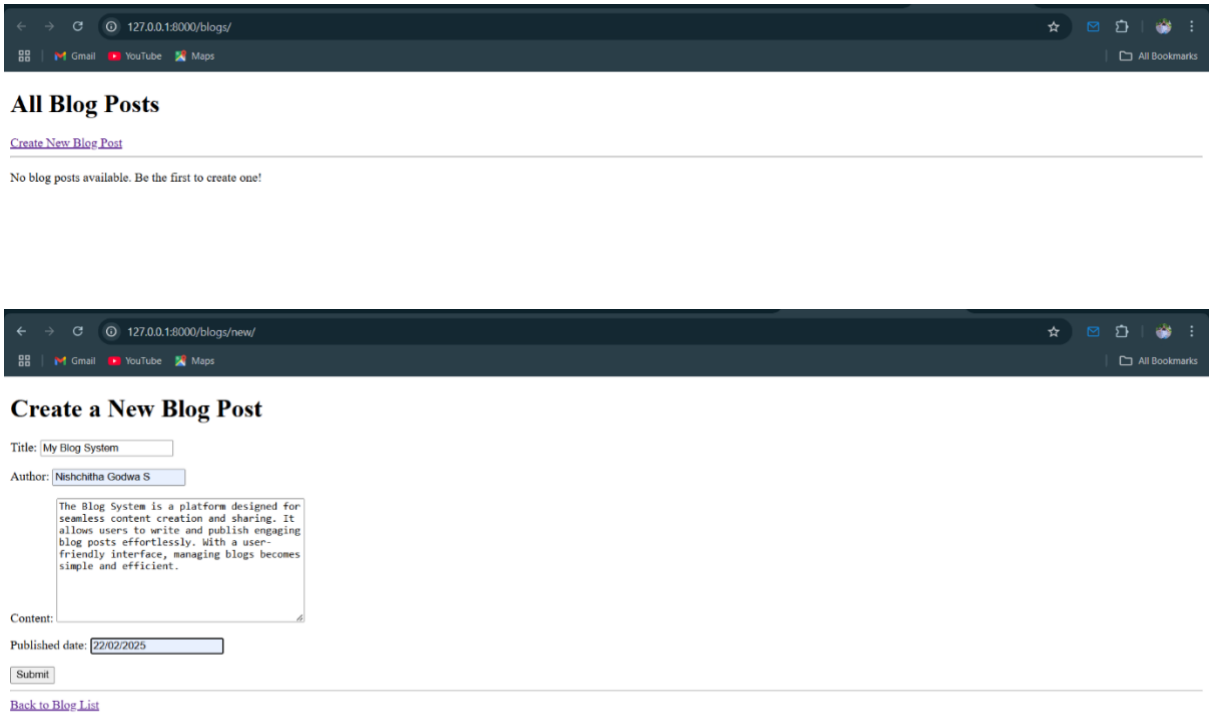
  python manage.py runserver

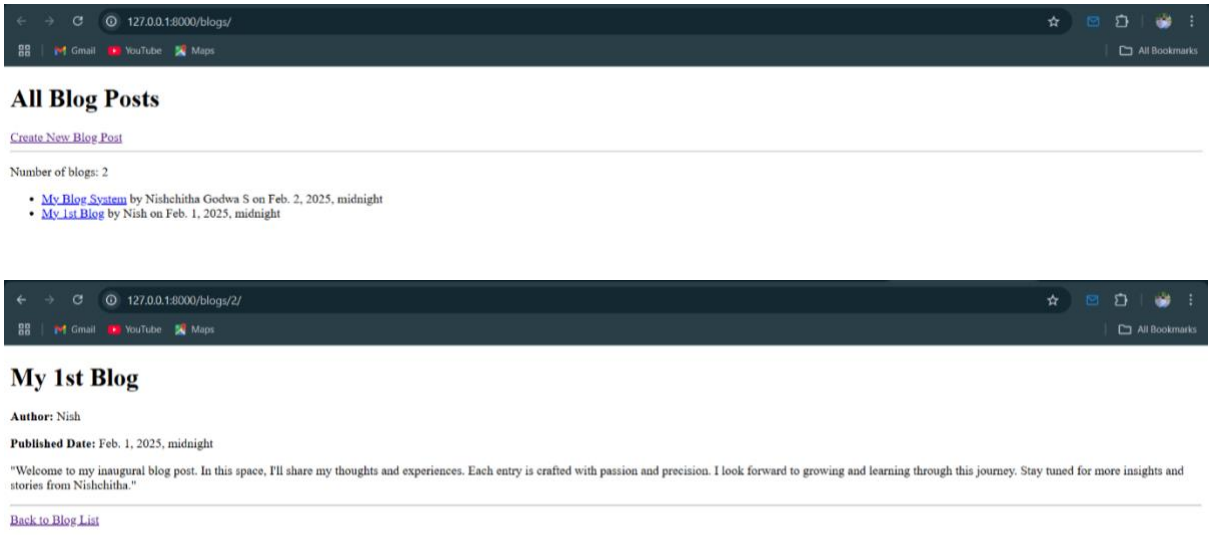
Conclusion:

You have successfully implemented a Django-based Blogging System with the following features:

- Users can create blog posts with a title, author, content, and published date.
- The system uses Django's generic CreateView, ListView, and DetailView.
- URL configurations include:
  - /blogs/ – List all blog posts.
  - /blogs/<int:pk>/ – Display a single blog post.
  - /blogs/new/ – Create a new blog post.
- After submission, users are redirected to the blog list page with a success message using reverse_lazy().

Output:





GitHub Link: https://github.com/NishhGowda/Multipage-Blog-System.git