Blog Post Recommendation using Django

Last Updated : 26 Dec, 2023

In this article, we will guide you through the creation of a blog post recommendation system using Django. Our article covers the integration of a user-friendly login system with a registration form, ensuring a seamless experience for your website visitors. Additionally, we have implemented a sophisticated search functionality that allows users to tailor their searches based on preferences and level ranges. This adaptive search system takes into account the varying levels of traffic, providing an efficient and personalized browsing experience for users exploring your blog.

Blog Post Recommendation using Django

Using this Blog Post Recommendation using the Django system user see the recommended blog according to their preference and also set by the traffic on the blog and also read the article by one simple click. To install Django follow these steps.

- Django Install
- Virtual Env

Starting the Project Folder

To start the project use this command

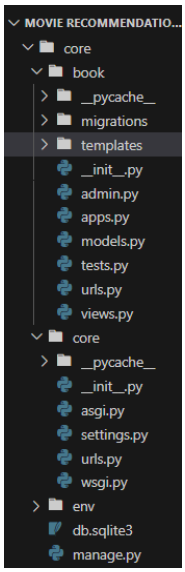
```
django-admin startproject core
cd core
python manage.py startapp book
```

Now add this app to the '**settings.py**'.

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
```

```
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "book",
]
```

File Structure



Setting Necessary Files

models.py: This Django code defines two models, "Emenitites" and "Blog." "Emenitites" has a name field, while "Blog" includes fields for blog name, description, price, and a ManyToMany relationship with "Emenitites." Both models use the "__str__" method to display their names, likely for representation in a web application.

Python3

```python
from django.db import models
from django.contrib.auth.models import User
# Create your models here.
class Emenitites(models.Model):
    name = models.CharField(max_length=100)
    def __str__(self):
        return self.name

```

```python
class Blog(models.Model):
    blog_name =models.CharField(max_length=100)
    blog_description = models.TextField()
    price = models.IntegerField()
    emenities = models.ManyToManyField(Emenitites)

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

views.py : This Django code defines several views. The "home" view renders a page displaying all amenities. The "api_blogs" view processes API requests, filtering blogs based on price and amenities. The "login_page" and "register_page" views handle user authentication and registration, displaying appropriate messages and redirecting users. The "@login_required" decorator ensures that only authenticated users can access the "home" and "api_blogs" views.

Python3

```python
from django.shortcuts import render, redirect
from .models import *
from django.http import JsonResponse
from django.contrib import messages
from django.contrib.auth import login, authenticate
from django.contrib.auth.decorators import login_required

# Create your views here.
@login_required(login_url="/login/")
def home(request):
    emenities = Emenitites.objects.all()
    context = {'emenities': emenities}
    return render(request, 'home.html', context)

@login_required(login_url="/login/")
def api_blogs(request):
    blogs_objs = Blog.objects.all()

    price = request.GET.get('price')
    if price :
        blogs_objs = blogs_objs.filter(price__lte=price)

    emenities = request.GET.get('emenities')
    if emenities:
        emenities = emenities.split(',')
        em = []
        for e in emenities:
            try:
                em.append(int(e))
            except Exception as e:
                pass
        # print(em)
        blogs_objs = blogs_objs.filter(emenities__in=em).distinct()
    payload = []
    for blog_obj in blogs_objs:
```

```
            result = {}
            result['blog_name'] = blog_obj.blog_name
            result['blog_description'] = blog_obj.blog_description
            result['price'] = blog_obj.price
            payload.append(result)

        return JsonResponse(payload, safe=False)

def login_page(request):
    if request.method == "POST":

        try:
            username = request.POST.get('username')
            password = request.POST.get('password')
            user_obj = User.objects.filter(username=username)
            if not user_obj.exists():
                messages.error(request, "Username not found")
                return redirect('/login/')
            user_obj = authenticate(username=username, password = passwc
            if user_obj:
                login(request, user_obj)
                return redirect('/')
            messages.error(request, "Wrong Password")
            return redirect('/login/')

        except Exception as e:
            messages.error(request, " Somthing went wrong")
            return redirect('/register/')

    return render(request, "login.html")


def register_page(request):
    if request.method == "POST":
        try:
            username = request.POST.get('username')
            password = request.POST.get('password')
            user_obj = User.objects.filter(username=username)
            if user_obj.exists():
                messages.error(request, "Username is taken")
                return redirect('/register/')
            user_obj = User.objects.create(username=username)
            user_obj.set_password(password)
            user_obj.save()
            messages.success(request, "Account created")
            return redirect('/login/')

        except Exception as e:
            messages.error(request, " Somthing went wrong")
            return redirect('/register/')

    return render(request, "register.html")
```

core/urls.py : This Django URL configuration file includes the main project's URL patterns. It routes the base URL to the 'book.urls' app, allowing for additional URL patterns to be defined in the 'book.urls' module. The 'admin/' path is also included for Django's admin interface.

Python3

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

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

book/urls.py: In this Django code snippet, URL patterns are defined for various views. The patterns include a home page (''), an API endpoint for blogs ('api/blogs'), and pages for user login ('login/') and registration ('register/'). These patterns are mapped to corresponding views imported from the 'views' module.

Python3

```python
from django.contrib import admin
from django.urls import path
from .views import *

urlpatterns = [
    path('', home, name='home'),
    path('api/blogs', api_blogs ),
    path('login/', login_page, name="login"),
    path('register/', register_page, name="register"),


]
```

Creating GUI

login.html: This HTML code is for a basic login page within a Bootstrap-styled container. It includes fields for a username and password, a “Login” button, and a link to create a new account. The page uses some custom CSS for styling and includes external libraries for additional styling.

Python3

```python
{% extends "base.html" %}

{% block start %}
<div class="container mt-5 mx-auto col-md-3 card shadow p-3" style="back
    <div class="login-form">
        {% if messages %}
            {% for message in messages %}
                <div class="alert alert-success {{ message.tags }} mt-4"
                    {{ message }}
                </div>
            {% endfor %}
        {% endif %}
        <form action="" method="post">
            {% csrf_token %}
            <h2 class="text-center" style="color: #333;">Log In</h2>
            <br>
```

```
            <div class="form-group">
                <input type="text" class="form-control" name="username"
              required style="background-color: #fff; border: 1px solid
            </div>
            <div class="form-group mt-3">
                <input type="password" class="form-control" name="passwo
              required style="background-color: #fff; border: 1px solid
            </div>
            <div class="form-group mt-4">
                <button class="btn btn-success btn-block" >Log In </butt
            </div>
        </form>
        <p class="text-center" style="color: #555;"><a href="{% url 'reg
        style="color: #007bff;">Registered Here</a></p>
    </div>
</div>
{% endblock %}
```

register.html: This HTML code is for a simple registration page. It includes fields for a username and password, a "Register" button, and a link to log in. The page uses Bootstrap and Font Awesome for styling and includes custom CSS for additional styling.

Python3

```
{% extends "base.html" %}

{% load static %}

{% block start %}

<div class="container mt-5 mx-auto col-md-3 card shadow p-3">
    <div class="login-form">
        {% if messages %}
            {% for message in messages %}
                <div class="alert alert-success {{ message.tags }}" role
                    {{ message }}
                </div>
            {% endfor %}
        {% endif %}
        <form action="" method="post">
            {% csrf_token %}
            <h2 class="text-center">Register</h2>
            <br>
            <div class="form-group">
                <input type="text" class="form-control" name="username"
            </div>
            <div class="form-group mt-3">
                <input type="password" class="form-control" name="passwo
            </div>
            <div class="form-group mt-3">
                <button class="btn btn-success btn-block">Register</butt
            </div>
        </form>
        <p class="text-center"><a href="{% url 'login' %}">Log In</a></p
    </div>
```

```
</div>

<style>
    /* Custom CSS for UI enhancements */
    .card {
        background-color: #f7f7f7;

    }

    .login-form {
        padding: 20px;
    }

    .btn-primary {
        background-color: #007bff;
        border-color: #007bff;
    }

    .btn-primary:hover {
        background-color: #0056b3;
        border-color: #0056b3;
    }


</style>

{% endblock %}
```

home.html : This HTML document creates a webpage for a blog post recommendation system using Django. It includes a navigation bar, styling with Materialize CSS and Font Awesome, and dynamic content loading. The page allows users to select blog types and filter by traffic using a range input. The JavaScript function "getBlogs()" fetches and displays recommended blogs based on user selections, with each blog presented in a card format.

Python3

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0
    <title>Django Blogs</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

    <!-- Add Materialize CSS -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/

    <!-- Add Materialize JavaScript (optional, if you need JavaScript fe
    <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
    <style>
        /* Custom CSS styles for your page */
        body {
            background-color: #f5f5f5;
        }
        .nav-wrapper {
```

```
        background-color: white ; /* Bootstrap's typical warning col
    }

    .brand-logo {
        margin-left: 20px;
        margin-top: -1%;

    }

    .container {
        margin-top: 20px;
    }

    .card {
        margin: 10px;
    }

    .card-title {
        font-size: 1.2rem;
        color: #333; /* Text color for card titles */
    }

    .range-field {
        padding: 20px 0;
    }

    /* Added styles for better spacing and alignment */
    .input-field {
        margin-bottom: 20px;
    }
    .gfg{
        margin-left: 40%;
        font-size: 40px;
       font-weight: 500;
        color: green;
    }
    /* Adjusted card width and added shadow */
    .card {
        width: 100%;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
    }

    /* Centered card content */
    .card-content {
        text-align: center;
    }
    #web{
        margin-left: 85%;
        font-size: 20px;
        padding: 5px 20px;
        background-color: rgb(101, 119, 225);
    }.JT{
        margin-top: -10%;
        font-size: 23px;
        color: black;
        font-weight: 400;
    }.ex{
```

```
            margin-top: 2%;
        }
        .blog{
            color: black;
        }
    </style>
</head>
<body>
    <nav>
        <div class="nav-wrapper">
            <a href="/" class="brand-logo"><h4 class="blog">Blog Post Re
            <a href="" id="web"> <i class="fas fa-globe"></i> Website</a
             </div>
    </nav>
    <h1 class="gfg"> GeeksforGeeks</h1>
    <br>
    <br>
    <div class="container">
        <div class="row">
            <div class="col m5">
                <div class="input-field col s12">
                    <select multiple  onchange="getBlogs()" id="emenitie
                                <option value="" disabled selected>Ch
                        {% for emenitie in emenities %}
                        <option value="{{emenitie.id}}">{{emenitie.name}
                        {% endfor %}
                    </select>
                    <label for="emenities"><h3 class="JT"> <i class="fas
                </div>
            </div>

            <div class="col m4 ex">
                <label for="price"><h3 class="JT"> Short By Traffic : </
                <p class="range-field">
                    <input type="range"  onchange="getBlogs()" id="price
                </p>
            </div>
        </div>
    </div>
    <div class="container">
        <div class="row" id="show_blogs_here">
        </div>
    </div>

    <script>
        // Initialize the show_blogs_here variable
        var show_blogs_here = document.getElementById("show_blogs_here")

        $(document).ready(function(){
            $('select').formSelect();
        });

        function getBlogs() {
            var price = document.getElementById("price").value;
            var instance = M.FormSelect.getInstance(document.getElementB
            var emenities = '';
```

```
            var html = '';

            if(instance){
                emenities = instance.getSelectedValues();
            }

            fetch(`/api/blogs?emenities=${emenities}&price=${price}`)
            .then(result => result.json())
            .then(response => {
                for (var i = 0; i < response.length; i++) {
                    // Use template literals (backticks) to create the H
                    html += `
                        <div class="col s12 m4 " >
                            <div class="card" id="blog">

                                <div class="card-content">
                                    <span class=" gfg1" >${response[i].t
                                    <p class ="gfgd" >${response[i].blog
                                    <p class = "ex23"> Traffic >: <stron
                                     <br>
                                    <button type="submit" class ="btn">
                                </div>
                            </div>
                        </div>
                    `;
                }
                show_blogs_here.innerHTML = html;
            });

        }
       getBlogs()
    </script>
    <style>
        #blog{
            border-radius: 10px;
            border: 0.5px solid black;
        }
        .ex23{
            font-size: 10px;
        }
        .gfg1{
            color: rgb(78, 71, 71);
            font-size: 25px;
            font-weight: bold;
        }.gfgd{
             color: gray;
        }
        .btn{
            padding: 0px 10px;
            font-weight: bold;
            background-color: rgb(226, 84, 84);
        }
    </style>
</body>
</html>
```

base.html: This HTML document is a template for a webpage using Django. It loads static files, includes Bootstrap and Font Awesome for styling, and creates a navigation bar with login and register links. The title is "Blog Post Recommendation," and it provides a block named "start" for content to be added in derived templates. The page also features a header with "GeeksforGeeks" and includes optional JavaScript with Bootstrap for additional functionality.

Python3

```
{% load static %}
<!doctype html>
<html lang="en">

<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/boot
    integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65V
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/fc


  <title>Blog Post Recommendation</title>
</head>
<body>
  <nav class="navbar navbar-light shadow-lg ">
    <div class="container-fluid">

      <a href="{% url 'home' %}" class="navbar-brand " style="color:blac
      <div class="d-flex">
        <a href="{% url 'login' %}" class="navbar-brand"><h4>Login</h4><
        <a href="{% url 'register' %}" class="navbar-brand"><h4>Register
      </div>
    </div>
  </nav>

  <h1 class="text-center" style="margin-top: 2%; color:green; font-size:
  {% block start %}{% endblock %}

  <!-- Optional JavaScript; choose one of the two! -->

  <!-- Option 1: Bootstrap Bundle with Popper -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/boot
    integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+
    crossorigin="anonymous"></script>

  <!-- Option 2: Separate Popper and Bootstrap JS -->
  <!--
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/
    integrity="sha384-IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgF
    crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bc
    integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0
    crossorigin="anonymous"></script>
    -->
```

```
48  </body>
49
50  </html>
51  </body>
52
53  </html>
```

admin.py: Here we are registering our models.

Python3

```
from django.contrib import admin
from book.models import *
# Register your models here.

admin.site.register(Blog)
admin.site.register(Emenitites)
```

Deployement of the Project

Run these commands to apply the migrations:

```
python3 manage.py makemigrations
python3 manage.py migrate
```

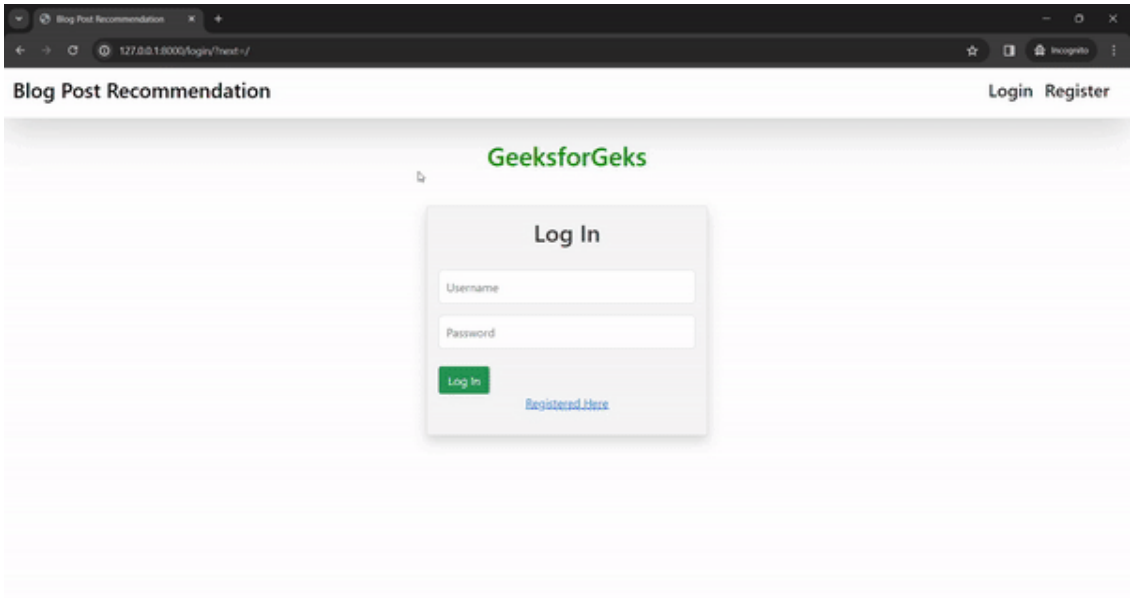
Create super user by the following command, and then create username, email and password.

```
python manage.py createsuperuser
```

Now go to **http://127.0.0.1:8000/admin** and data to the database.
Run the server with the help of following command:

```
python3 manage.py runserver
```

Output



Previous Article

Online Auction System Using Django

Next Article

Similar Reads

Movie Recommendation System using Django

In this article, we will guide you through the process of creating a comprehensive Movie Recommendation System with the added functionality of user authentication. By...

8 min read

Building Blog CMS (Content Management System) with Django

Django is a python based web application framework that is helpful for building a variety of web applications. Django also includes an extensible Django-Admin interface, Defau...

10 min read

Restaurant Recommendation App using MEAN

In this article, we'll explore the process of building a restaurant recommendation application using Angular for the client side and Node.js with Express for the server sid...

10 min read

Restaurant Recommendation using MERN

This article is about Restaurant Recommendations using the MERN (MongoDB, Express.js, React.js, Node.js) stack. This project displays a list of restaurants to the user...

9 min read

Book Recommendation System using Node and Express.js

The Book Recommendation System aims to enhance the user's reading experience by suggesting books tailored to their interests and preferences. Leveraging the power of...

4 min read

Project Idea | Songs Recommendation System in Android

Project Title: Songs Recommendation System in Android Introduction: We all know that in today's era internet is expanding very much and as a result, the data, as well as other...

2 min read

Project Idea | Recommendation System based on Graph Database

The main objective of this project is to build an efficient recommendation engine based on graph database(Neo4j). The system aims to be a one stop destination for...

1 min read

Movie Recommendation System with Node and Express.js

Building a movie recommendation system with Node and Express will help you create personalized suggestions and recommendations according to the genre you selected. T...

3 min read

Movie recommendation based on emotion in Python

Movies that effectively portray and explore emotions resonate deeply with audiences because they tap into our own emotional experiences and vulnerabilities. A well-crafte...

4 min read

Create a Blog App using React-Native

This article shows how to create a basic blog app using react native. This app contains functionalities such as adding a blog and a delete button to remove the blogs using rea...

4 min read

Article Tags :
Django
Geeks Premier League
Project
Geeks Premier League 2023

+1 More

Corporate & Communications Address:- A-143, 9th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305) | Registered Address:- K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305