

BLOG WEBSITE USING DJANGO

NAME	VIGNESH S
PROJECT NAME	BLOG WEBSITE
PROGRAMMING LANGUAGE	PYTHON (Django Framework)
INSTITUTION NAME	BESANT TECHNOLOGIES

OBJECTIVES

My motive is to create User Friendly Blog Website to Create an intuitive and easy-to-navigate platform for users to view, create, and interact with blog posts and can access the post and content dynamically.

Primary Goals

- ❖ Enable CRUD operations (Create, Read, Update, Delete) for posts
- ❖ Provide an easy-to-use interface for creating and viewing blog posts.
- ❖ Implement user authentication for posting and commenting.
- ❖ Include an admin interface for managing users, posts, and comments, ensuring easy content moderation
- ❖ Allow users to update and personalize their profiles, enhancing the user experience and engagement.

TECHNOLOGY STACK

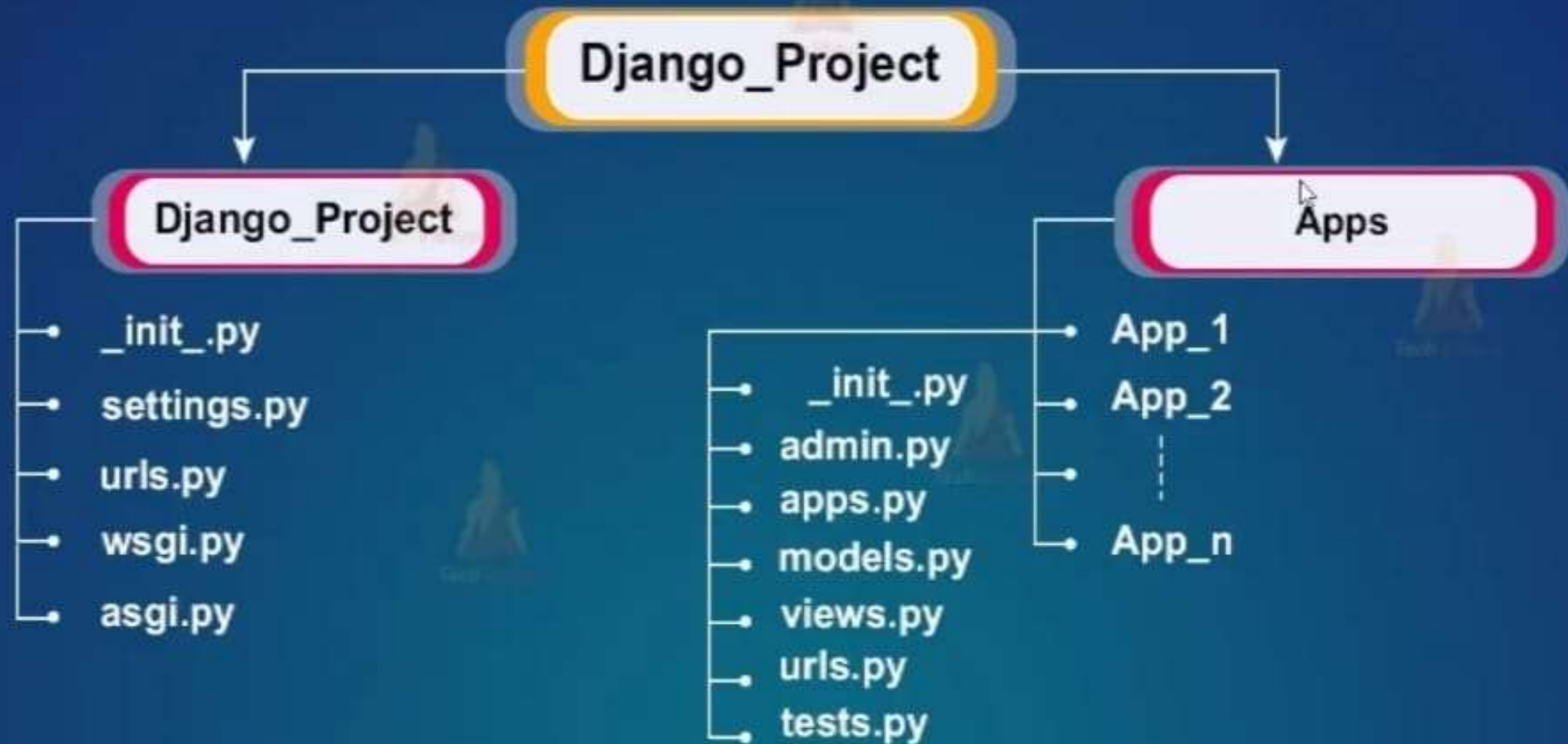
- **Backend** : Django, Python
- **Frontend** : HTML, CSS
- **Database** : MYSQL
- **Tools & Libraries**
 1. Mysqlclient for connecting with Mysql database
 2. Faker to generate demo data
 3. Django-Pagination for Page layout
 4. Pillow for image handling
 5. Django-rest-framework for API interface

INSTALLATIONS STEPS OF DJANGO

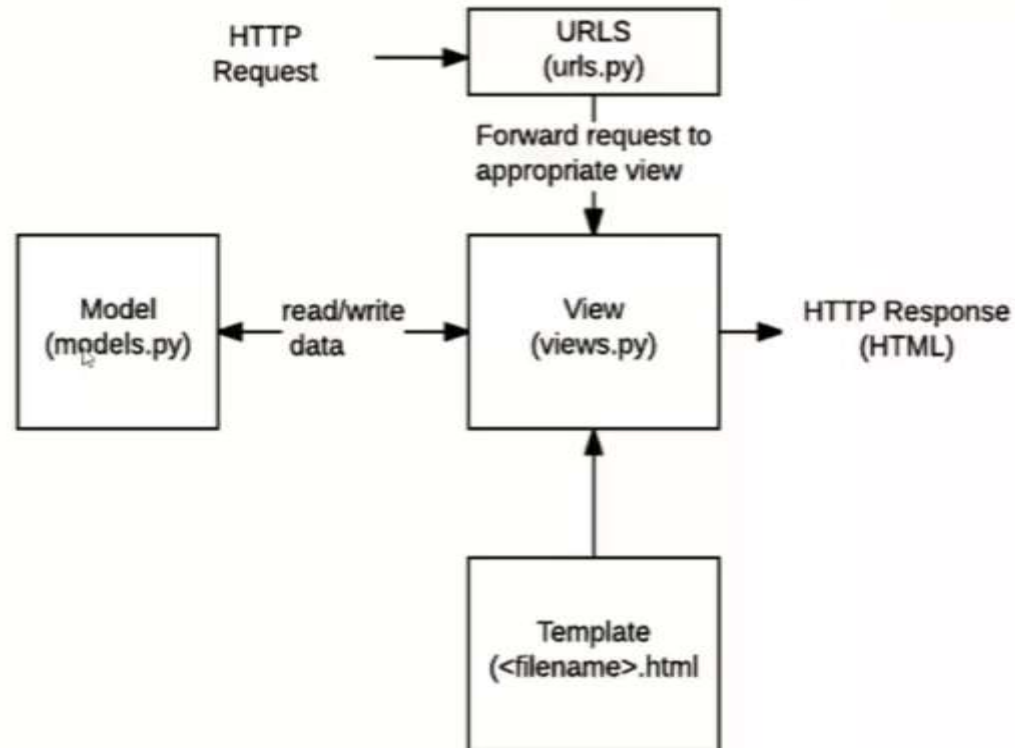
- **Set Up Virtual Environment:** `python -m venv env` and activate it.
- **Install Django:** `pip install django`
- **Install Django rest-framework :** `pip install Django-rest-framework`
- **Install Mysqlclient :** `pip install mysqlclient`
- **Verify Django:** `django-admin --version`.
- **Create Project:** `django-admin startproject myproject ..`
- **Create App:** `python manage.py startapp APP`
- **Apply Migrations:** Set up database schema using `python manage.py migrate`.
- **Create Superuser:** Optional step for admin access via `python manage.py createsuperuser`.
- **Run Server:** Start development server with `python manage.py runserver` and access via browser.



Django File Structure



EXECUTION FLOW



REDIRECT AND REVERSE URL

➤ **Redirect url:**

```
from django.shortcuts import redirect

def old_url_redirect(request):
    return redirect("new_url")
def new_url_view(request):
    return HttpResponse('This is new url')
```

➤ **Reverse function:**

```
from django.urls import reverse

def old_url_redirect(request):
    return redirect(reverse('blog:new_url'))
```

VARIBALE INTERPOLATION AND FILTERS

❖ Variable Interpolation:

Variables from the view context are displayed in templates using double curly braces `{{ }}`.

For example, `{{ user.name }}` displays the name attribute of the user object.

❖ Filters:

Filters are applied to variables within templates to format or modify them.

Syntax: `{{ variable|filter_name }}`.

For example, `{{ date|date:"Y-m-d" }}` formats a date.

date: Formats date objects (e.g., `{{ my_date|date:"d M Y" }}`).

length: Returns the length of a list or string.

default: Sets a default value if the variable is undefined.

truncatechars: It returns based on the mentioned characters

❖ Custom Filters:

You can create custom filters in Django by defining a function and registering it in template.

These tools together enable dynamic and user-friendly templates, making data display customizable directly from the template layer.

FOR TAG AND IF TAG

```
{% if posts %}  
  {% for post in posts %}  
    <div class="col-4 mb-4"> </div>  
  {% endfor %}  
  {% else %}  
    <p>No Posts Available</p>  
  {% endif %}
```

CREATING MODELS

```
from django.db import models
from django.utils.text import slugify
# Posts
class Post(models.Model):
    title=models.CharField(max_length=100)
    content=models.TextField()
    img_url=models.URLField(blank=True)
    created_at=models.DateTimeField(auto_now_add=True)
    slug=models.SlugField(unique=True)
    category=models.ForeignKey(Category,on_delete=models.CASCADE)
    def save(self,*args,**kwargs):
        if not self.slug:
            self.slug=slugify(self.title)
            super().save(*args,**kwargs)
```

CREATING MIGRATIONS

```
from django.db import migrations, models
class Migration(migrations.Migration):
    dependencies = [
        ('blog', '0002_post_created_at'),
    ]
    operations = [
        migrations.AddField(
            model_name='post',
            name='slug',
            field=models.SlugField(default='example-slug', unique=True),
            preserve_default=False,
        ),
    ]
```

HANDLING ERROR AND CUSTOMIZE PAGE

```
from django.http import Http404,HttpResponseForbidden
def detail(request,slug):
    try:
        post=Post.objects.get(slug=slug)
        related_posts=Post.objects.filter(category=post.category).exclude(pk=post.id)
    except Post.DoesNotExist:
        raise Http404("Post does not exists")
def __call__(self, request):
    if request.user.is_authenticated and not request.user.has_permission:
        return HttpResponseForbidden("You do not have permission to access this page")
    response = self.get_response(request) return response
```

GENERATE SLUG FOR POSTS

```
from django.db import models
from django.utils.text import slugify

class Post(models.Model):
    title=models.CharField(max_length=100)
    content=models.TextField()
    img_url=models.URLField(blank=True)
    created_at=models.DateTimeField(auto_now_add=True)
    slug=models.SlugField(unique=True)
    category=models.ForeignKey(Category,on_delete=models.CASCADE) //many to one relationship

    def save(self,*args,**kwargs):    //to generate slug
        if not self.slug:
            self.slug=slugify(self.title)
            super().save(*args,**kwargs)
```

PAGINATION

-- Terminal

Pip install Django-pagination

-- views.py

```
from django.core.paginator import Paginator
```

```
def index(request):
```

```
    blog_title='New Posts'
```

```
    # getting data from post model
```

```
    all_posts=Post.objects.all()
```

```
    paginator=Paginator(all_posts,5)
```

```
    page_number=request.GET.get('page')
```

```
    page_obj=paginator.get_page(page_number)
```

```
    return render(request,'index.html',{ 'blog_title':blog_title,'page_obj':page_obj})
```

ENABLE CSRF COOKIE IN FORMS

1. Add CSRF Middleware

```
# settings.py MIDDLEWARE = [ ... 'django.middleware.csrf.CsrfViewMiddleware', ... ]
```

2. Use csrf_token :

```
<form method="POST" action="/submit-form/">
    {% csrf_token %}
    <label for="name">Name:</label>
    <input type="text" name="name" id="name" required>
    <button type="submit">Submit</button>
</form>
```

3. Configure settings.py

```
# settings.py
```

```
CSRF_COOKIE_HTTPONLY = True
```

ADMIN SETUP

-- Terminal

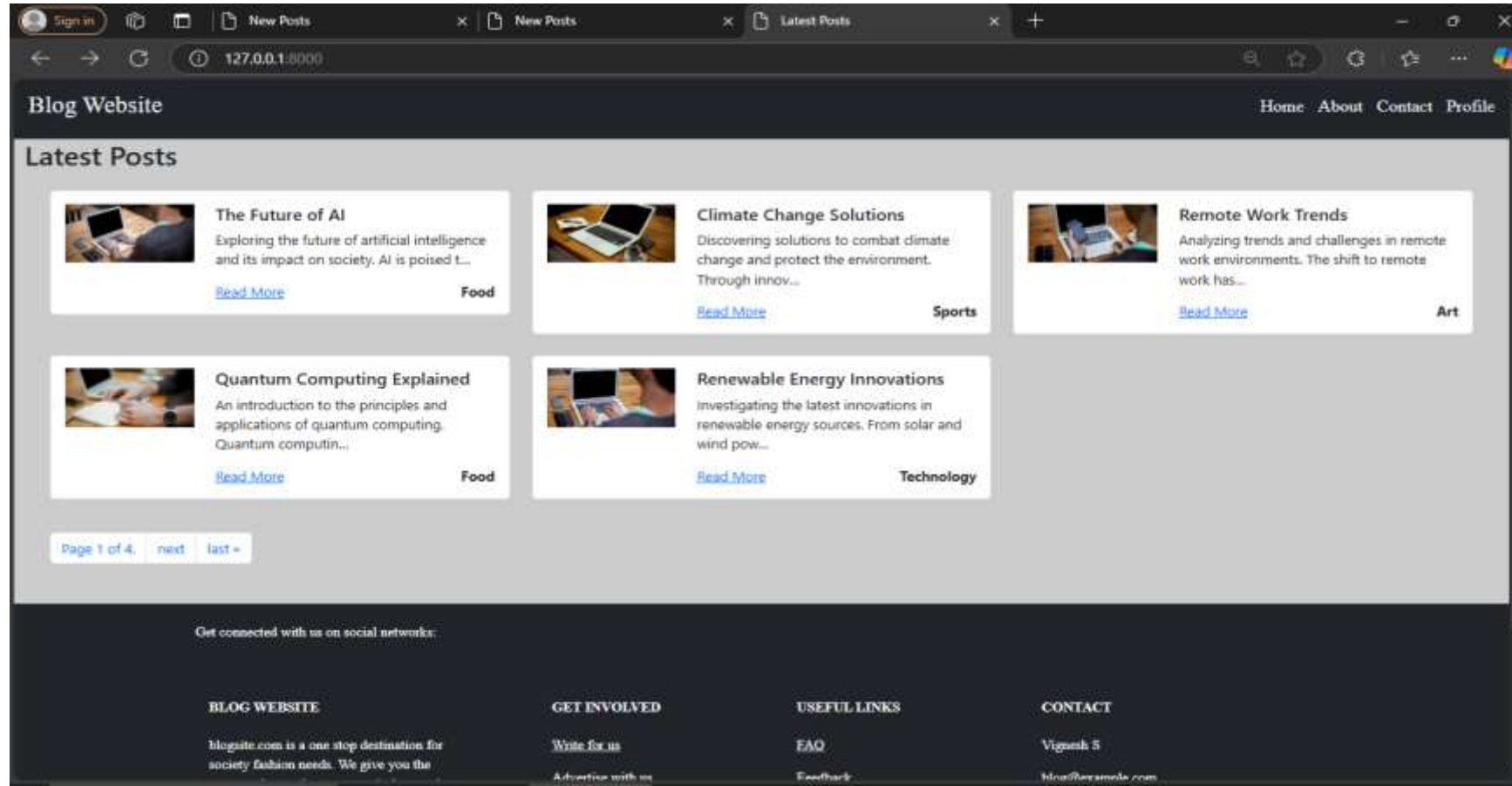
Python manage.py createsuperuser

-- admins.py

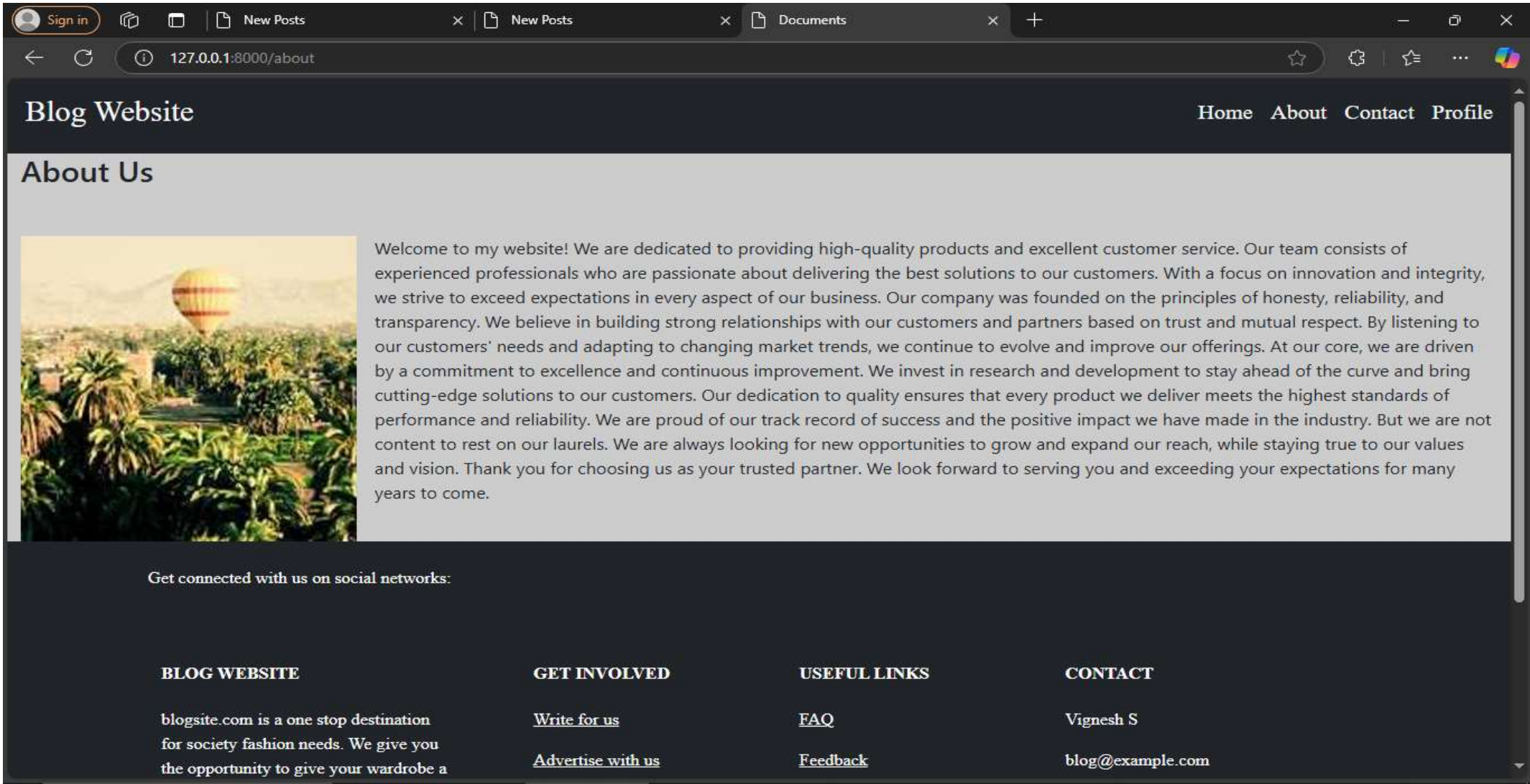
```
from django.contrib import admin
from .models import Post,Category,Aboutus
class PostAdmin(admin.ModelAdmin):
    list_display=('title','content')
    search_fields=('title','content')
    list_filter=('category','created_at')
# Register your models here.
admin.site.register(Post,PostAdmin)
admin.site.register(Category)
admin.site.register(Aboutus)
```


BLOG WEBSITE OUTPUT

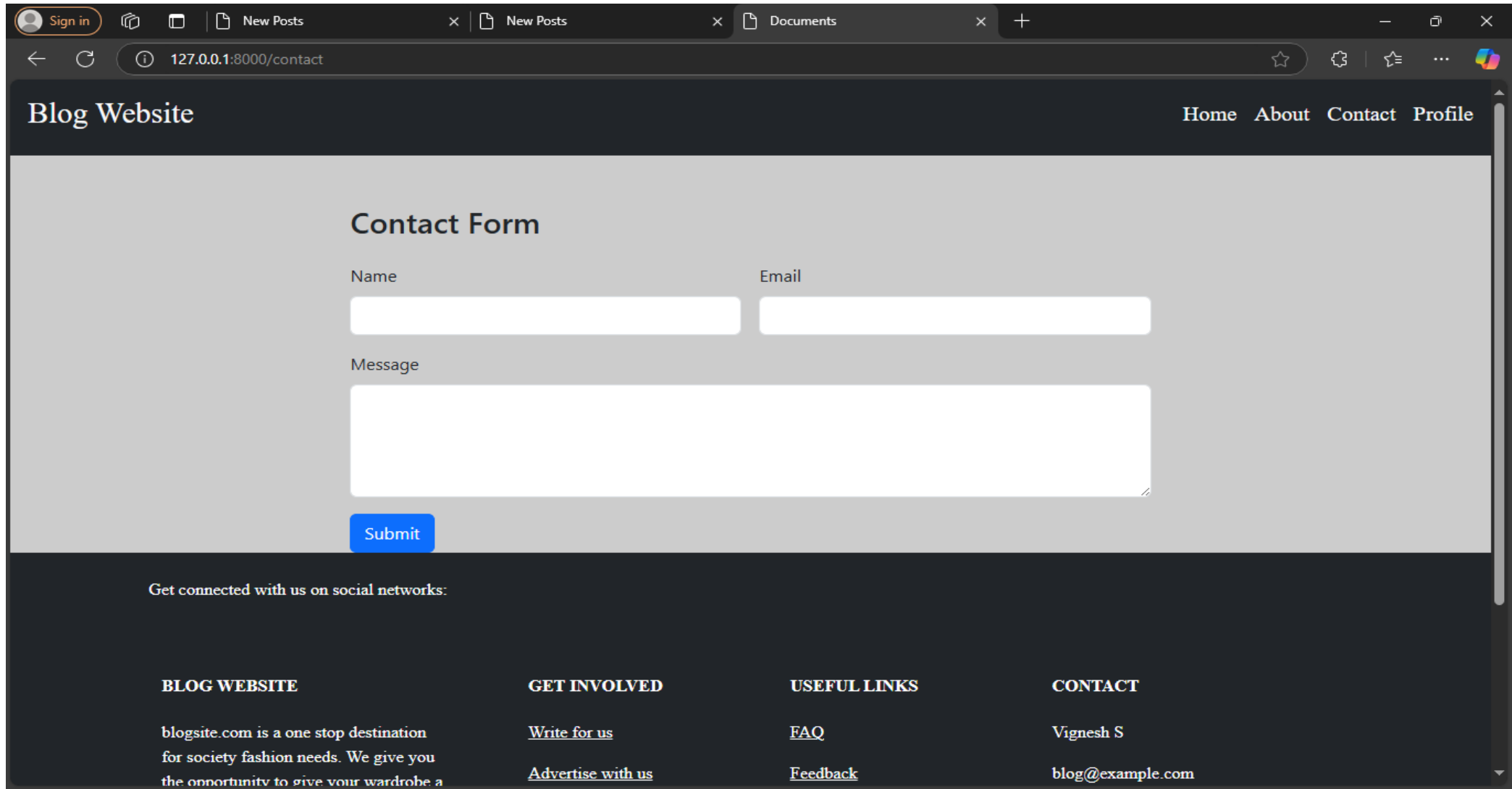
Home Page



About us Page



Contact Form Page



The screenshot shows a web browser window with three tabs: 'New Posts', 'New Posts', and 'Documents'. The address bar displays '127.0.0.1:8000/contact'. The website has a dark header with 'Blog Website' on the left and navigation links 'Home', 'About', 'Contact', and 'Profile' on the right. The main content area is light gray and contains a 'Contact Form' with three input fields: 'Name', 'Email', and 'Message'. A blue 'Submit' button is located below the 'Message' field. Below the form, a dark footer section contains the text 'Get connected with us on social networks:' followed by four columns of links and contact information.

Blog Website Home About Contact Profile

Contact Form

Name Email

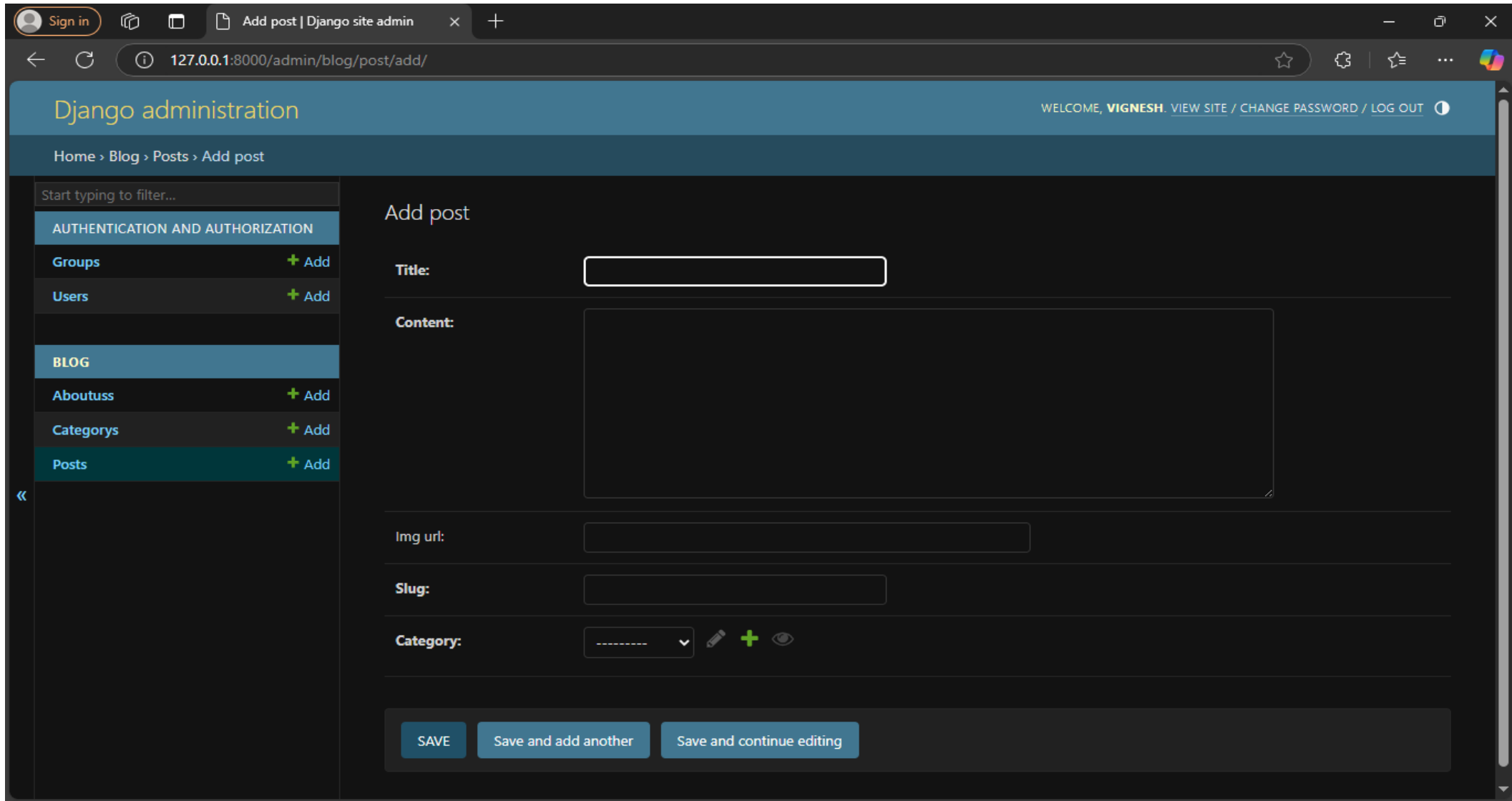
Message

Submit

Get connected with us on social networks:

BLOG WEBSITE	GET INVOLVED	USEFUL LINKS	CONTACT
blogsite.com is a one stop destination for society fashion needs. We give you the opportunity to give your wardrobe a	Write for us Advertise with us	FAQ Feedback	Vignesh S blog@example.com

Admin Site



The screenshot displays the Django administration interface in a web browser. The browser's address bar shows the URL `127.0.0.1:8000/admin/blog/post/add/`. The page header includes the Django logo and the text "Django administration", along with a welcome message for "VIGNESH" and links to "VIEW SITE", "CHANGE PASSWORD", and "LOG OUT". The breadcrumb trail indicates the current location: "Home > Blog > Posts > Add post".

The left sidebar contains a search bar and a list of menu items. Under "AUTHENTICATION AND AUTHORIZATION", there are links for "Groups" and "Users", each with a "+ Add" button. Under the "BLOG" section, there are links for "Aboutuss", "Categorys", and "Posts", each also with a "+ Add" button. The "Posts" link is currently selected and highlighted.

The main content area is titled "Add post" and contains the following form fields:

- Title:** A text input field.
- Content:** A large text area for the post content.
- Img url:** A text input field for the image URL.
- Slug:** A text input field for the post slug.
- Category:** A dropdown menu with a placeholder "-----", a pencil icon for editing, a green plus icon for adding a new category, and an eye icon for toggling visibility.

At the bottom of the form, there are three buttons: "SAVE", "Save and add another", and "Save and continue editing".



Thank You All :)