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

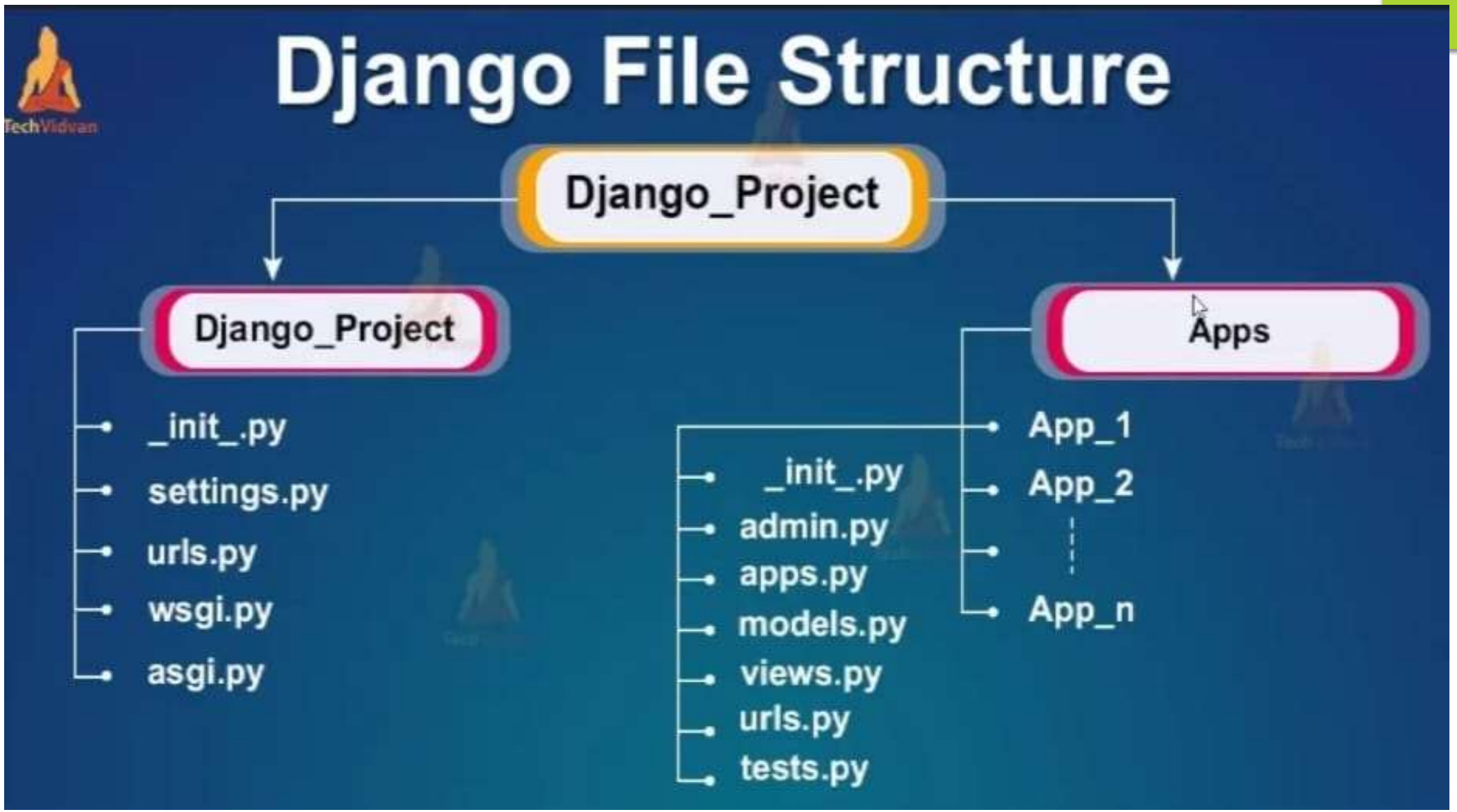
- Enable CRUD operations (Create, Read, Update, Delete) for posts
- Provide an easy-to-use interface for creating and viewing blog posts.
- Implement user authentication for posting and commenting.
- Include an admin interface for managing users, posts, and comments, ensuring easy content moderation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

TECHNOLOGY STACK

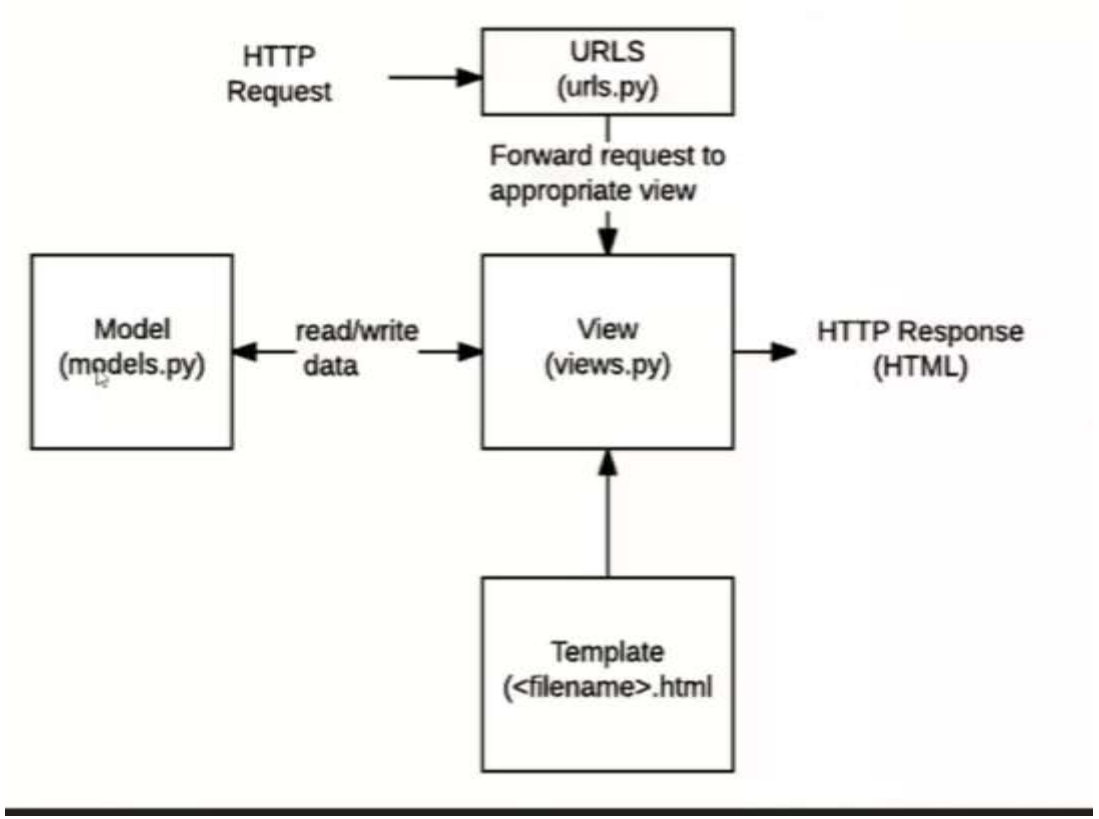
- **Backend** : Django, Python
- **Frontend** : HTML,CSS
- **Database** : MYSQL
- **Tools & Libraries**
    1. Mysqlclient for connecting with Mysql database
    2. Faker to generate demo data
    3. Django-Pagination for Page layout
    4. Pillow for image handling
    5. Django-rest-framework for API 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
TechVidvan
Django_Project
Django_Project
init.py
settings.py
urls.py
wsgi.py
asgi.py
Apps
App_1
App_2
App_n
init.py
admin.py
apps.py
models.py
views.py
urls.py
tests.py

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

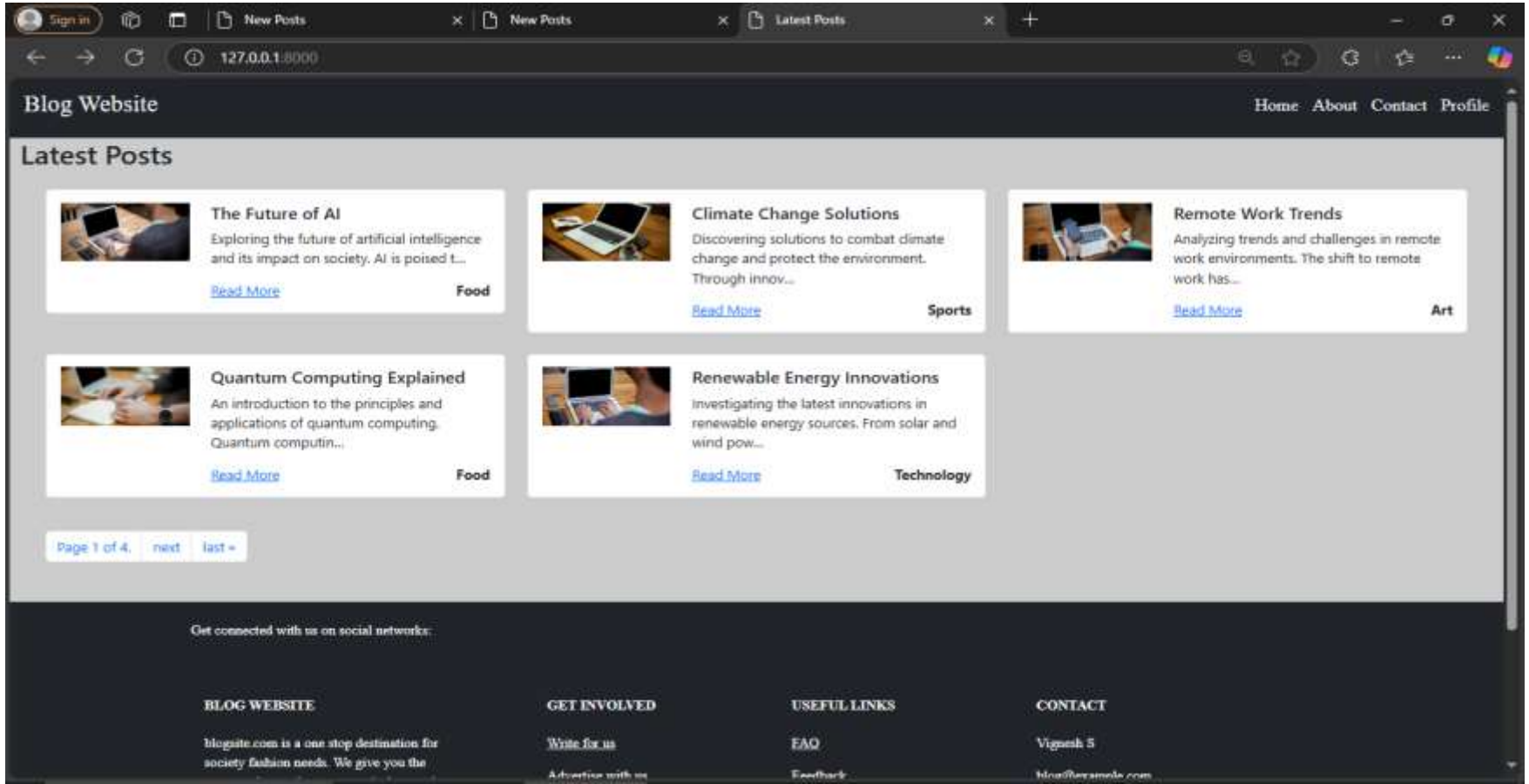
```
Python manage.py createsuperuser
```

-- admins.py

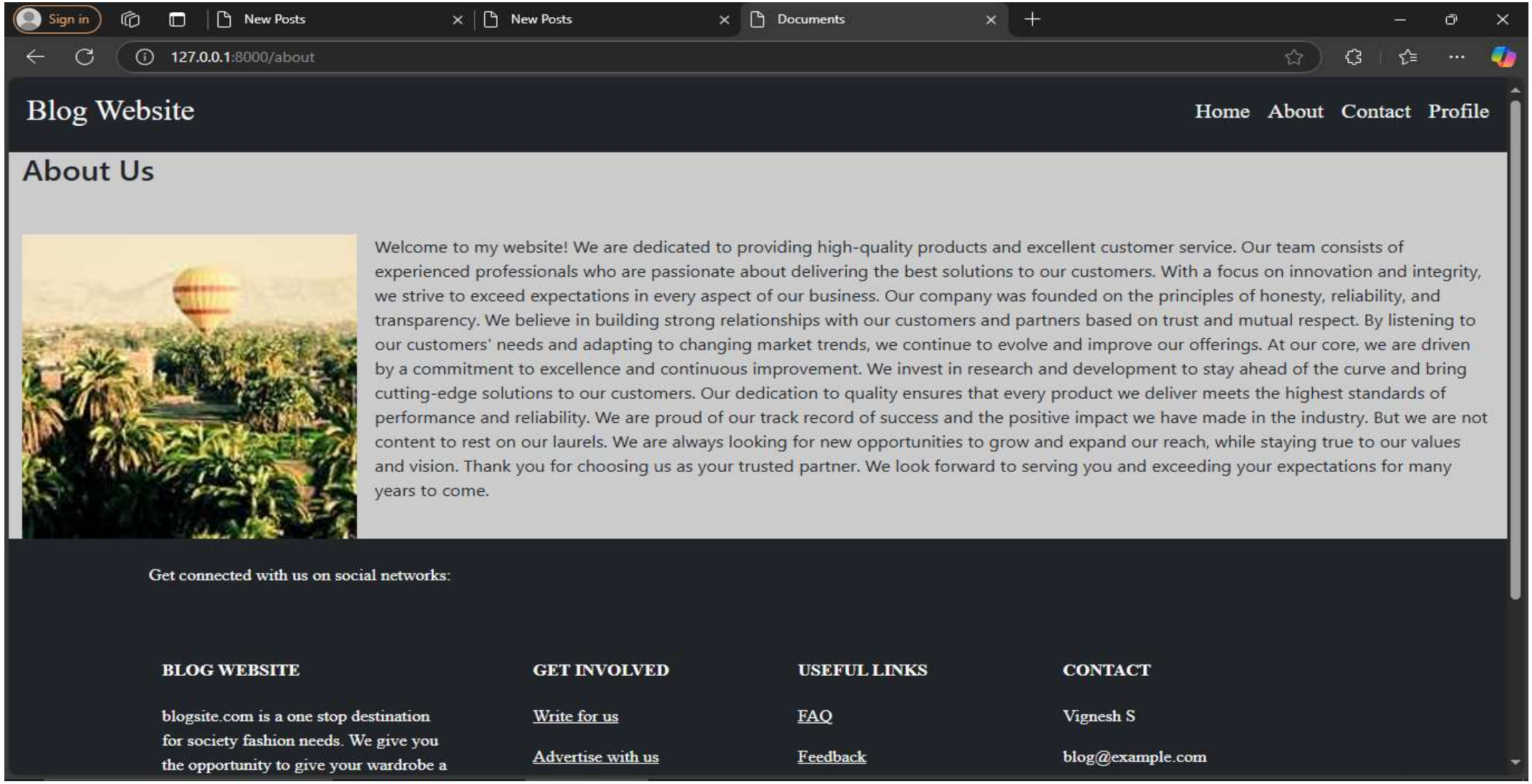
```
from django.contrib import admin
from .models import Post,Category,Aboutus
class PostAdmin(admin.ModelAdmin):
    list_display=('title','content')
    search_fields=('title','content')
    list_filter=('category','created_at')
# Register your models here.
admin.site.register(Post,PostAdmin)
admin.site.register(Category)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

BLOG WEBSITE OUTPUT

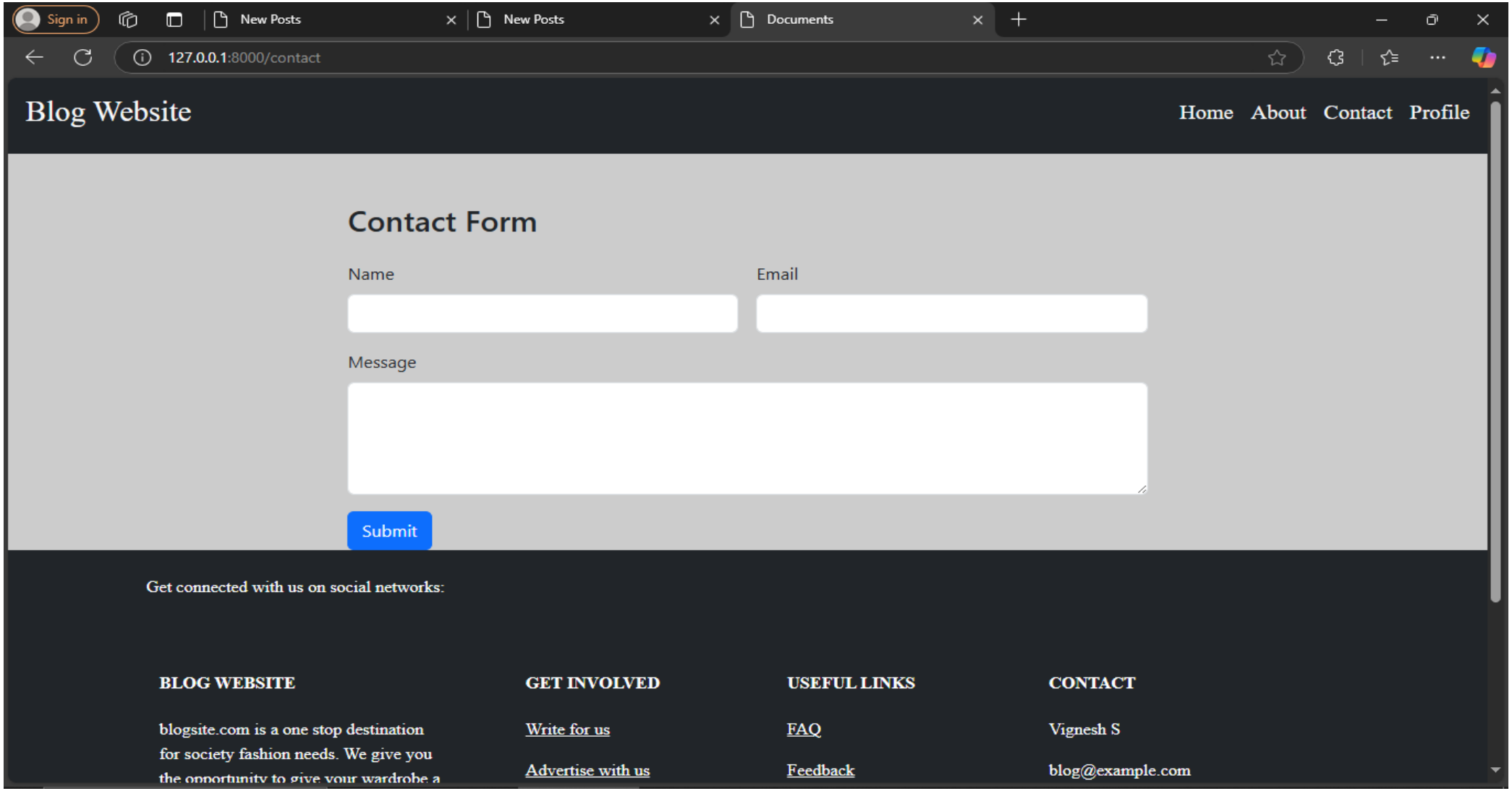
Home Page



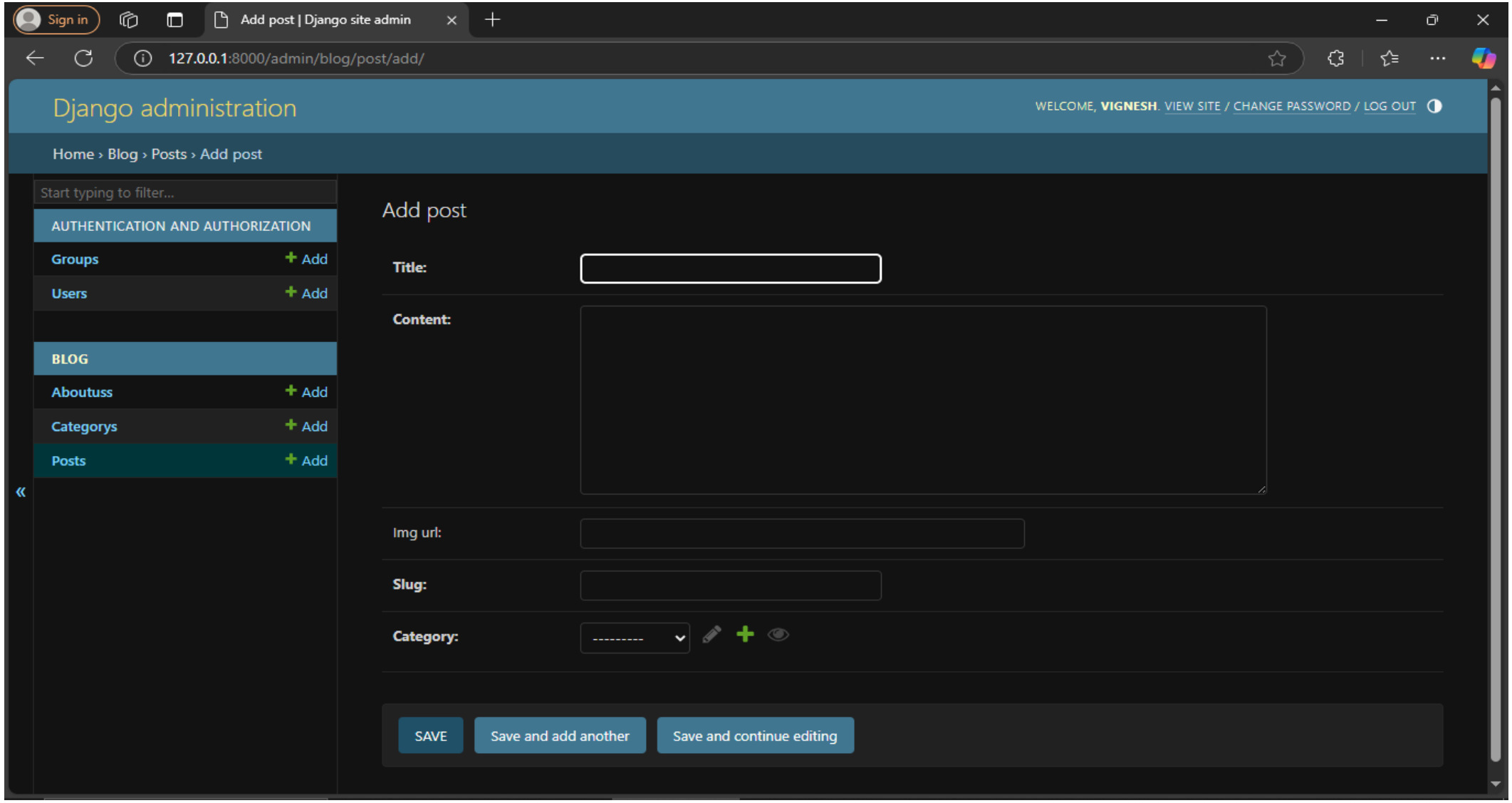
About us Page



Contact Form Page



Admin Site


Sign in
Add post | Django site admin
127.0.0.1:8000/admin/blog/post/add/
Django administration
WELCOME, VIGNESH. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Blog › Posts › Add post
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
BLOG
Aboutuss + Add
Categorys + Add
Posts + Add
Add post
Title:
Content:
Img url:
Slug:
Category:

SAVE
Save and add another
Save and continue editing

Thank You All :)