FULL STACK DEVELOPMENT PROJECT REPORT

TOPIC :- BLOG WEBSITE

Abhishek A	**1CD21IS004**
Dhanusha K S	**1CD21IS034**
Diwan G	**1CD21IS038**
Harish T S	**1CD21IS051**

Under the Guidance of

Mrs. Shoma

Asst. Professor Dept. of ISE

Citech, Bangalore

Introduction

Creating a blog website using Django and leveraging its administration interface offers a powerful way to manage your content. Django's admin interface, which is automatically generated from your models, allows you to easily handle CRUD (Create, Read, Update, Delete) operations on your blog's content. This interface is designed to be intuitive, providing a user-friendly experience even for those with limited technical expertise. With just a few lines of code, you can customize the admin interface to suit your needs, adding search fields, filters, and even inline editing for related models. This makes content management efficient and streamlined, allowing you to focus on creating quality blog posts without worrying about the technical details.

To get started, you need to set up your Django project and define your blog models, such as Post, Category, and Comment, in your `models.py` file. By registering these models in the `admin.py` file, Django will automatically generate an admin interface for them. This interface enables you to add new posts, categorize them, and moderate comments directly from the admin panel. You can also customize the appearance and behaviour of the admin interface by using model admin classes. For example, you can define list displays to show specific fields, add filters for easy navigation, and use search fields to quickly locate content. Django's admin interface not only saves development time but also provides a powerful tool for managing your blog efficiently.

Project Objectives

1. **Efficient Content Management**: Use Django's admin interface to enable easy creation, editing, and deletion of blog posts, categories, and comments.

2. **Robust and Secure Backend**: Implement Django’s security features and authentication to ensure a reliable and protected backend.

3. **Customizable and Responsive Frontend**: Create a visually appealing and adaptable frontend using Django’s templating system.

4. **Enhanced User Engagement**: Integrate search functionality, commenting, and social media sharing to improve user interaction and engagement.

Software Configuration

Frontend: HTML, CSS

HTML - HTML is used to create the structure of the blog's frontend, defining elements like headers, paragraphs, images, and links. In Django, HTML templates are employed to dynamically render content by integrating data from the backend. These templates are enhanced with Django template tags to handle logic and display dynamic content.

CSS - Controls the visual presentation of the website. It defines styles like fonts, colours, layouts, and animations, ensuring the Blog website platform is visually appealing and user-friendly.

HTML and CSS are fundamental for building and styling the Blog Website interface. They enable the creation of interactive elements and a seamless user experience.

Backend: Django

This refers to the server-side framework used to develop the logic and functionality of the Blog Website platform. Django is a robust Python web framework that provides tools for:

1.**Handling User Requests**: In the blog project, Django handles user requests by mapping URLs to view functions which process the request, interact with the database, and return the appropriate HTML response.

2.**Processing Data**: Django processes data by using models to interact with the database, performing queries, updates, and deletions as needed based on user interactions and requests.

3.**Interacting with the Database**: In the blog project, Django interacts with the database using ORM (Object-Relational Mapping) to seamlessly perform CRUD operations through defined models.

Database: SQLite

In the blog project, SQLite is used as the database backend, allowing Django to store and manage blog post data, categories, and comments efficiently through its lightweight, file-based database system. It provides a simple setup with minimal configuration, making it ideal for development and small-scale deployments. Additionally, SQLite's integration with Django's ORM ensures smooth data handling and queries without requiring separate database management tools.

In the blog project, SQLite is used as the database backend, allowing Django to store and manage blog post data, categories, and comments efficiently through its lightweight, file-based database system. It provides a simple setup with minimal configuration, making it ideal for development and small-scale deployments. Additionally, SQLite's integration with Django's ORM ensures smooth data handling and queries without requiring separate database management tools. For production environments or larger-scale applications, SQLite can be easily swapped out for more robust databases like PostgreSQL or MySQL, ensuring that the project can scale and meet evolving performance needs.

Code

Models.py #blogs/models.py

```
from django.db import models
class Category(models.Model):
    name = models.CharField(max_length=30)

    class Meta:
        verbose_name_plural = "categories"

    def __str__(self):
        return self.name

class Post(models.Model):
    title = models.CharField(max_length=255)
    body = models.TextField()
    created_on = models.DateTimeField(auto_now_add=True)
    last_modified = models.DateTimeField(auto_now=True)
    categories = models.ManyToManyField("Category", related_name="posts")

    def __str__(self):
        return self.title

class Comment(models.Model):
    author = models.CharField(max_length=60)
    body = models.TextField()
    created_on = models.DateTimeField(auto_now_add=True)
    post = models.ForeignKey("Post", on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.author} on '{self.post}'"
```

Views.py

```
# blog/views.py
from django.http import HttpResponseRedirect
from django.shortcuts import render
from blog.models import Post, Comment
from blog.forms import CommentForm

def blog_index(request):
    posts = Post.objects.all().order_by("-created_on")
    context = {
        "posts": posts,
    }
    return render(request, "blog/index.html", context)

def blog_category(request, category):
    posts = Post.objects.filter(
        categories__name__contains=category
    ).order_by("-created_on")
    context = {
        "category": category,
        "posts": posts,
    }
    return render(request, "blog/category.html", context)

def blog_detail(request, pk):
    post = Post.objects.get(pk=pk)
    form = CommentForm()
    if request.method == "POST":
        form = CommentForm(request.POST)
        if form.is_valid():
            comment = Comment(
```

```
            author=form.cleaned_data["author"],
            body=form.cleaned_data["body"],
            post=post,
        )
        comment.save()
        return HttpResponseRedirect(request.path_info)


    comments = Comment.objects.filter(post=post)
    context = {
        "post": post,
        "comments": comments,
        "form": CommentForm(),
    }
    return render(request, "blog/detail.html", context)
```

Forms.py # blog/forms.py

```
from django import forms
class CommentForm(forms.Form):
    author = forms.CharField(
        max_length=60,
        widget=forms.TextInput(
            attrs={"class": "form-control", "placeholder": "Your Name"}
        ),
    )
    body = forms.CharField(
        widget=forms.Textarea(
            attrs={"class": "form-control", "placeholder": "Leave a comment!"}
        )
```

```
    )
```

Admin.py # blog/admin.py

```
from django.contrib import admin
from blog.models import Category, Comment, Post

class CategoryAdmin(admin.ModelAdmin):
    pass

class PostAdmin(admin.ModelAdmin):
    pass

class CommentAdmin(admin.ModelAdmin):
    pass
admin.site.register(Category, CategoryAdmin)
admin.site.register(Post, PostAdmin)
admin.site.register(Comment, CommentAdmin)
```

Urls.py# blog/urls.py

```
from django.urls import path
from . import views
urlpatterns = [
    path("", views.blog_index, name="blog_index"),
    path("post/<int:pk>/", views.blog_detail, name="blog_detail"),
    path("category/<category>/", views.blog_category, name="blog_category"),
]
```

SNAPSHOTS

Fig 1 – Home Page

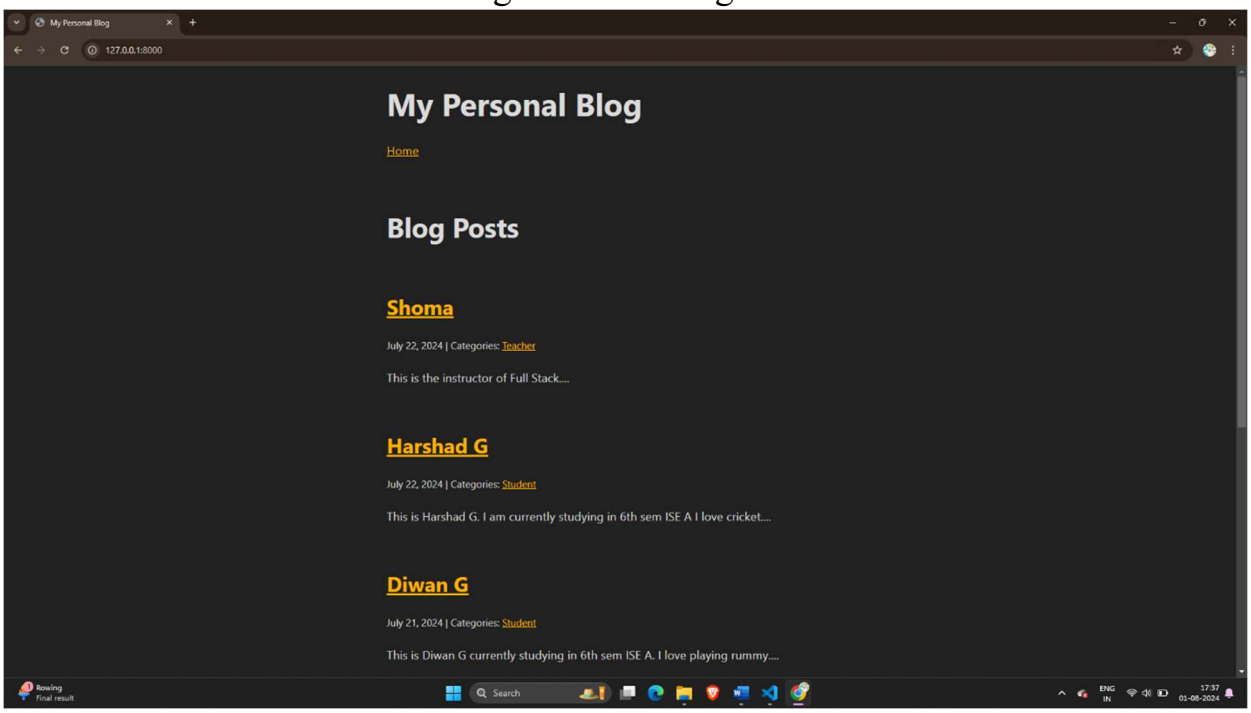
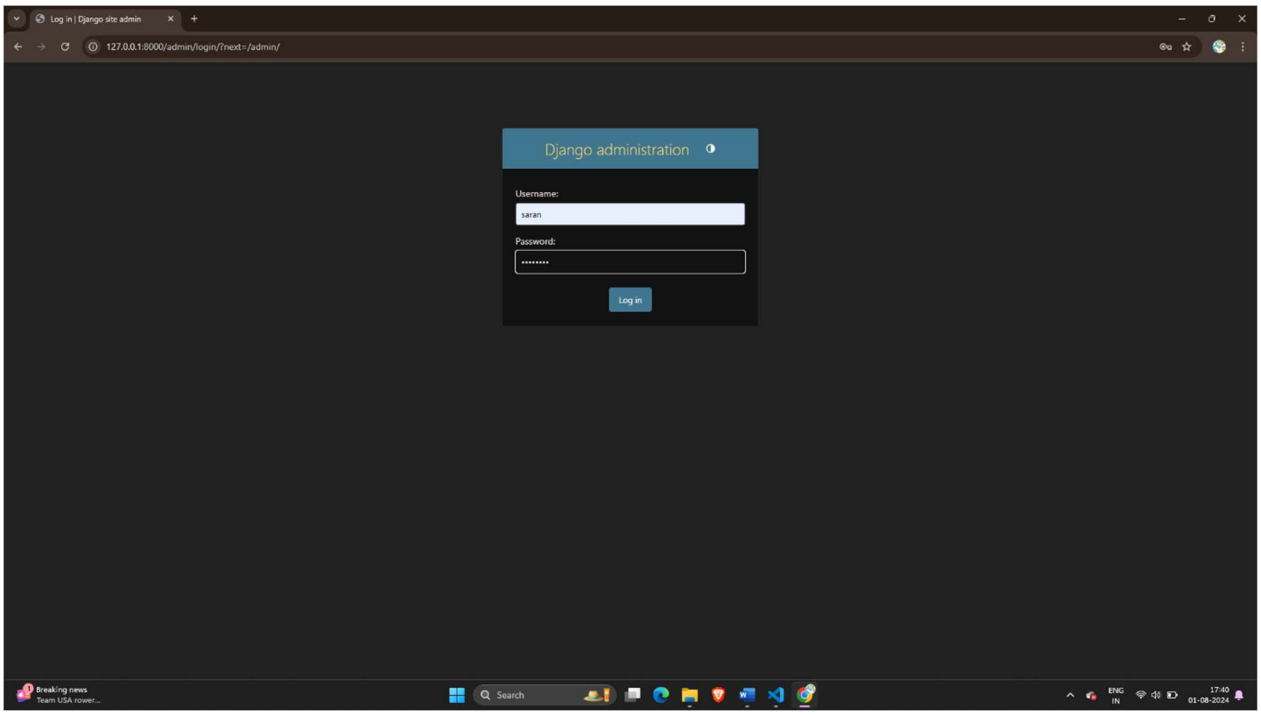


Fig 2- Admin Registration



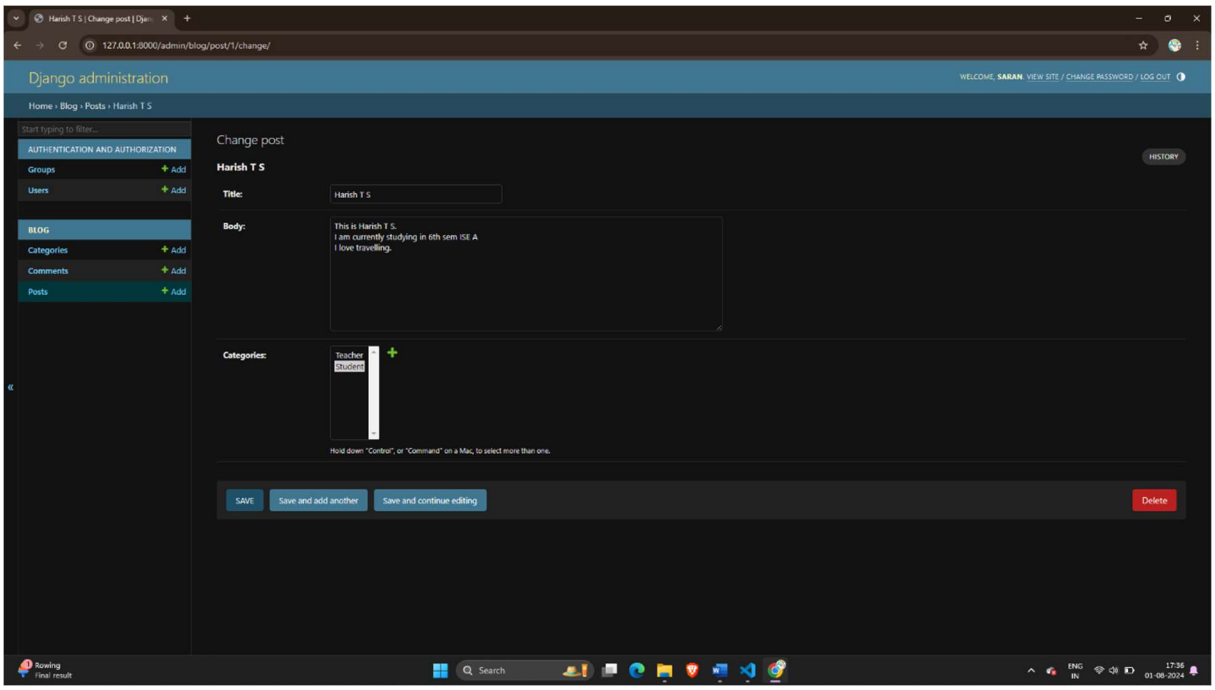


Fig 3 – Creation of new Post

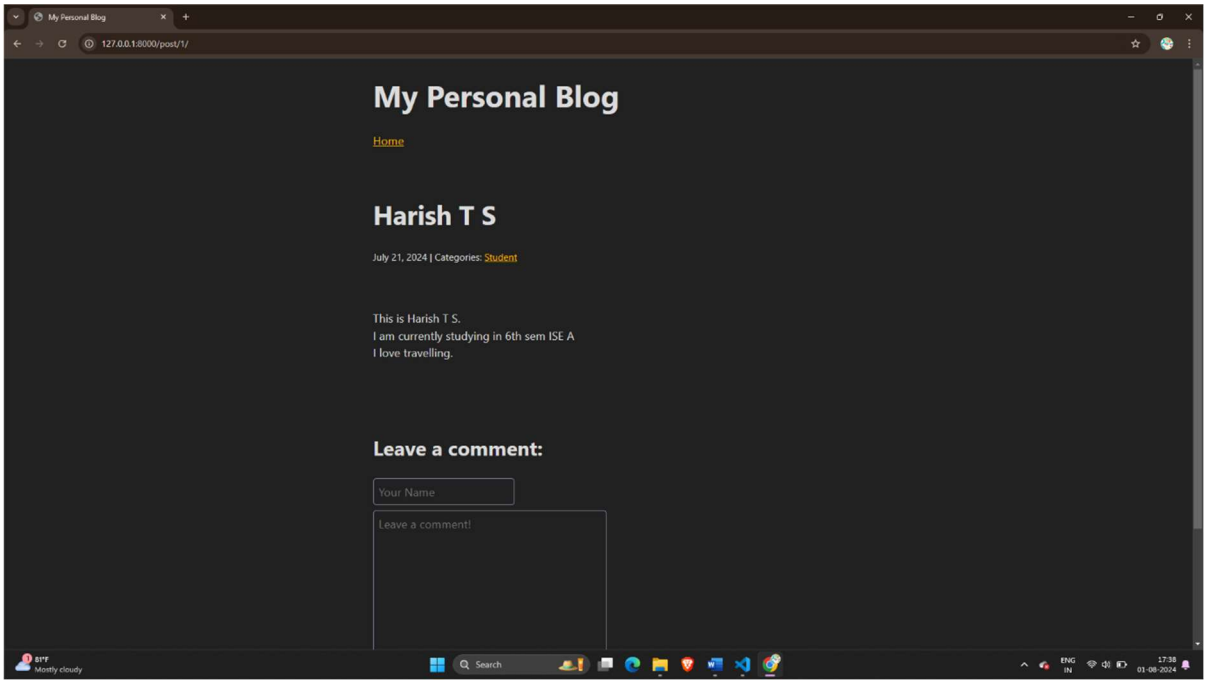


Fig 4 – Post created

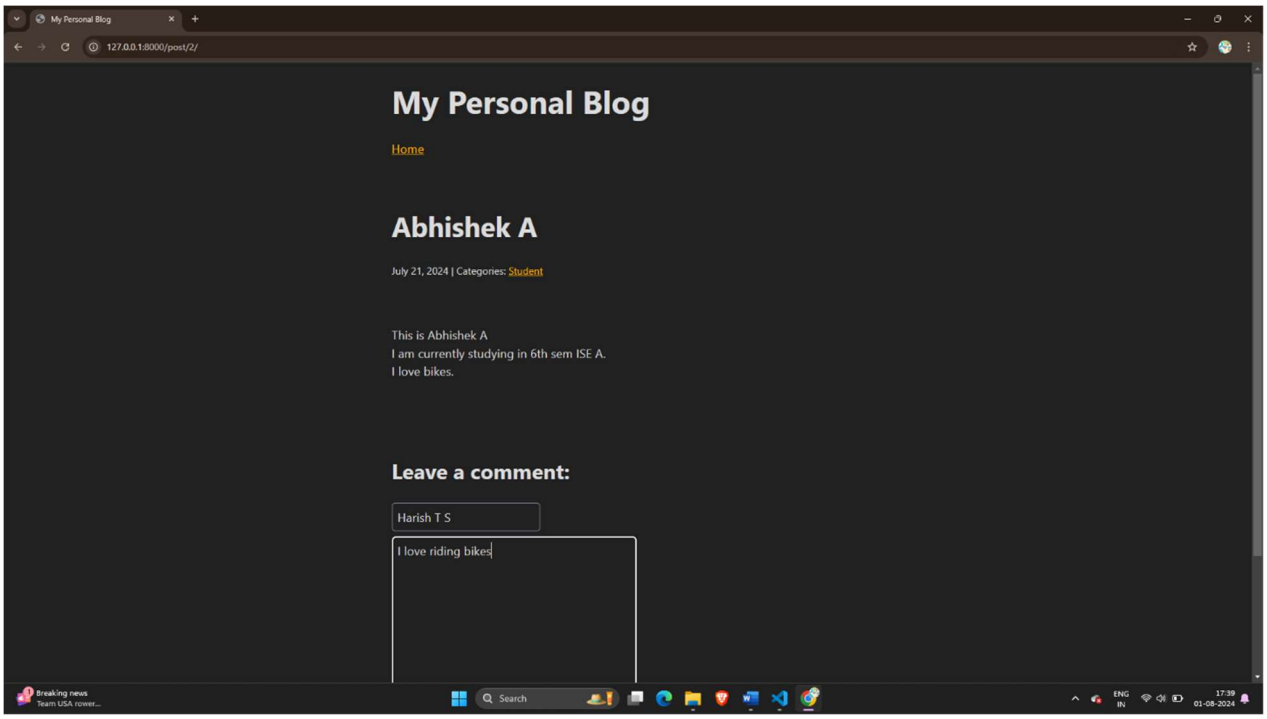


Fig 5 – Commenting on a post

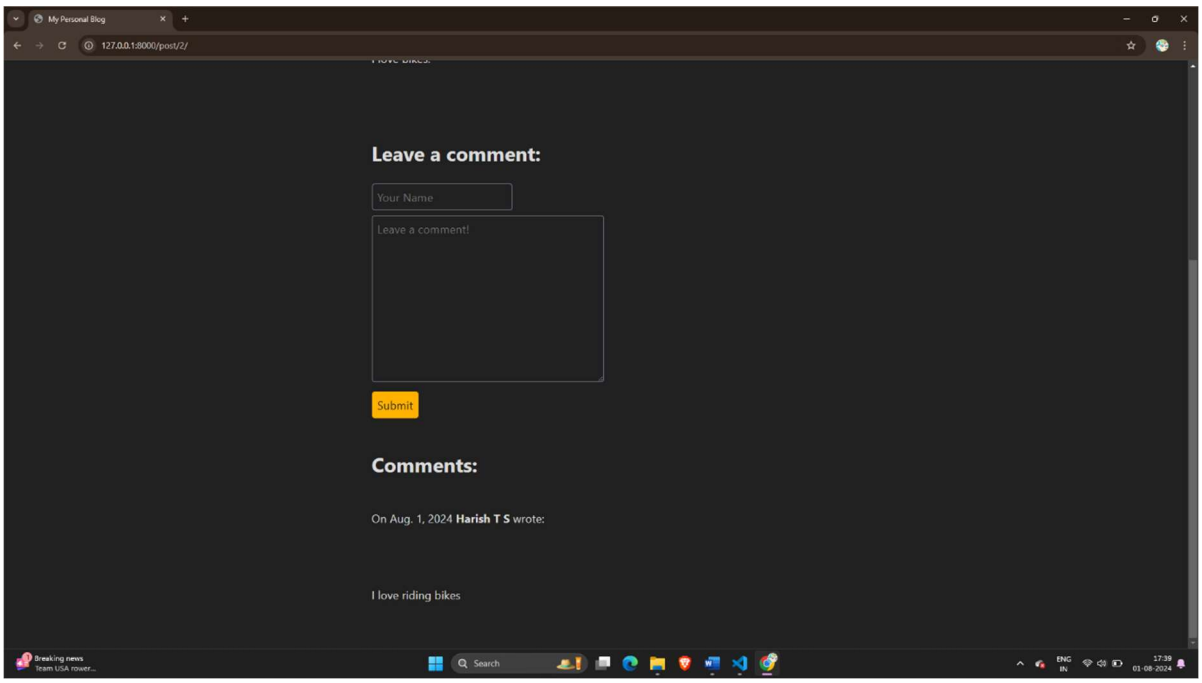


Fig 6– Comment section

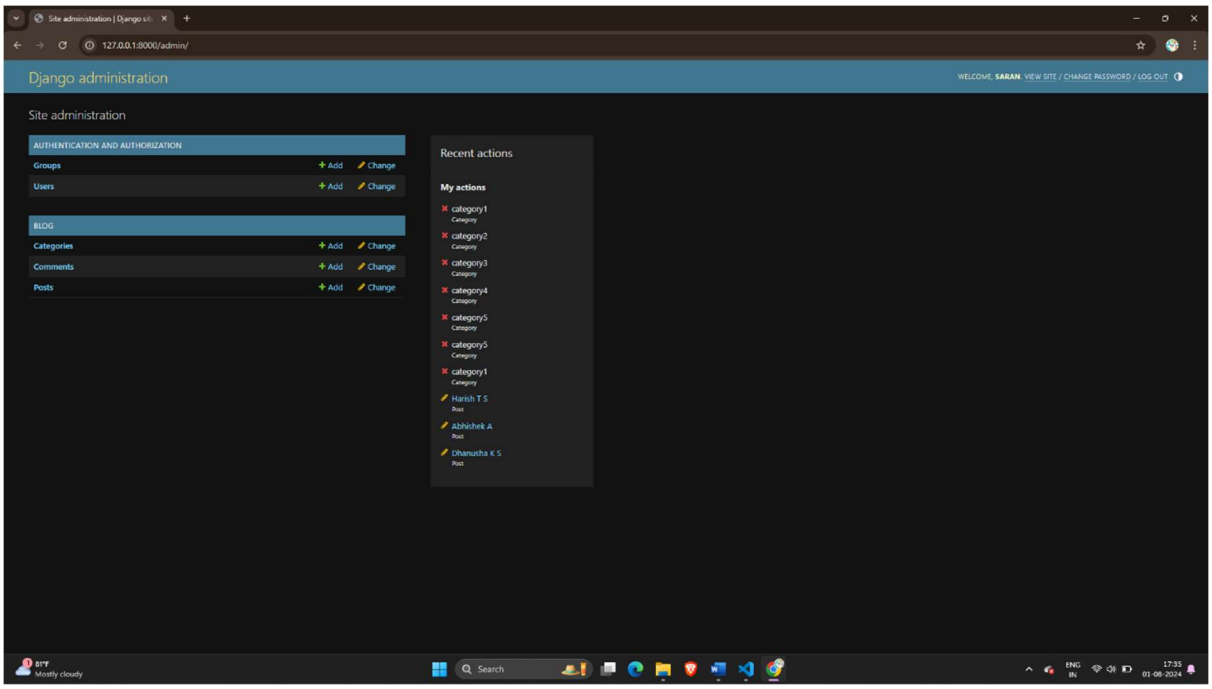


Fig 7 – Backend managing post, categories, comments

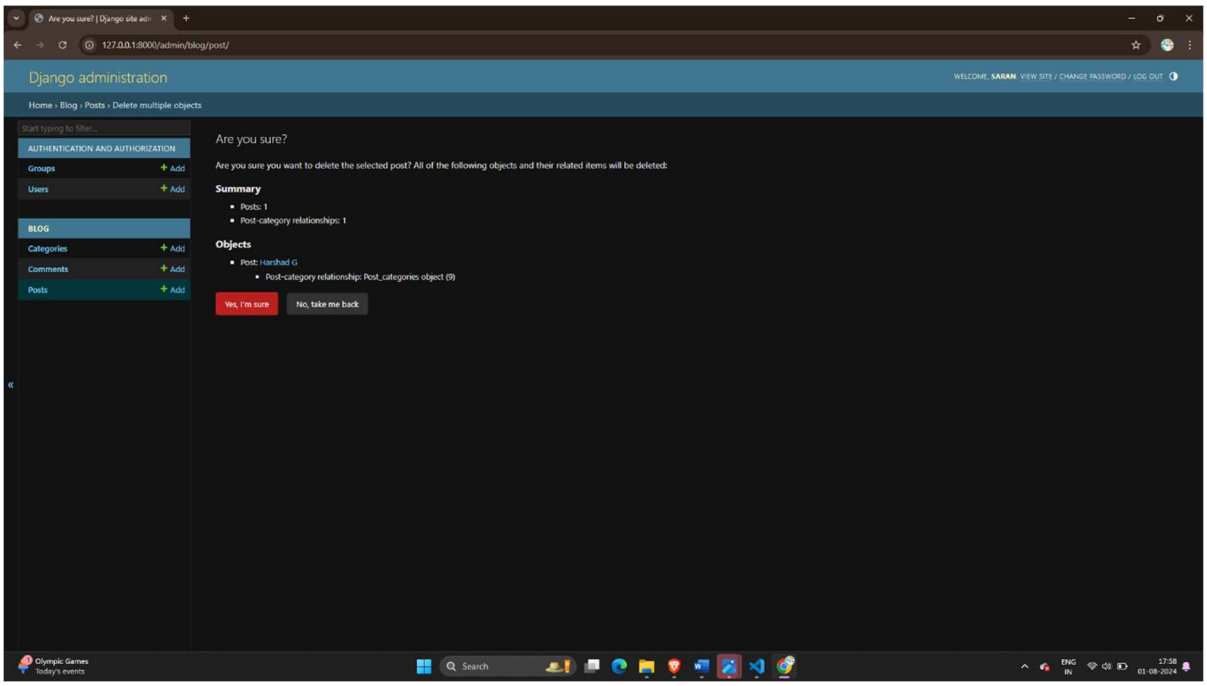


Fig 8 – Deletion of Post

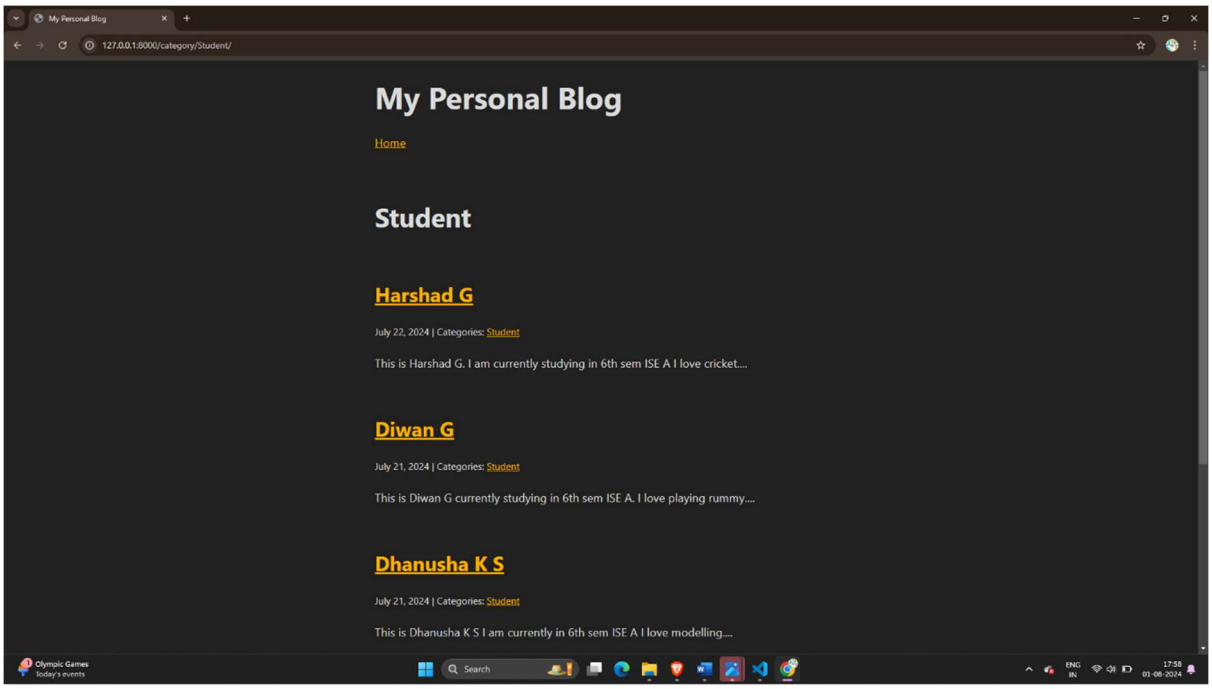


Fig 9 – Filtering of data on categories

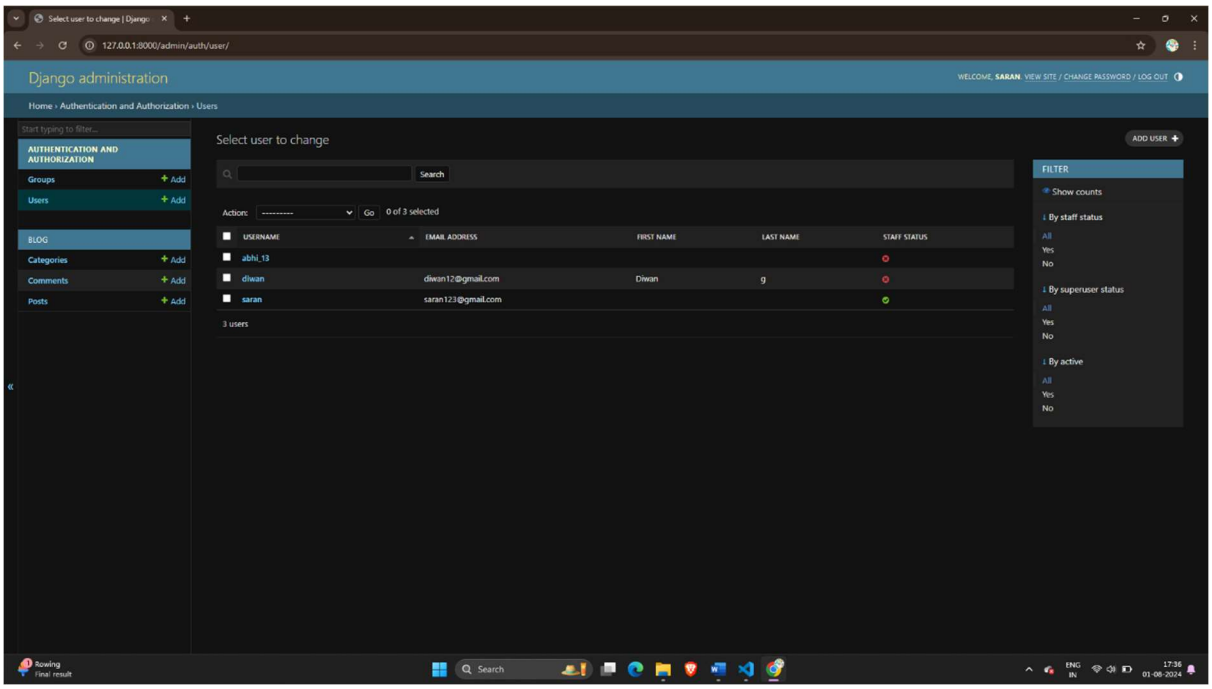


Fig 10 – Managing the users by superuser