QUICKDROP

QuickDrop: A File Sharing Hub

Submitted for the course: Internet and Web Programming (CSE3002)
in partial fulfilment of the requirements for the degree of

Bachelor of Technology
in
Computer Science and Engineering

by

SHAURYA CHOUDHARY
18BCE2113

Under the guidance of

Prof. Malathy E
SITE
VIT, Vellore.

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

May, 2021

DECLARATION

I hereby declare that the report entitled "QuickDrop: A File Sharing Hub" submitted by Shaurya Choudhary, for the award of the degree of *Bachelor of Technology in CSE* to VIT is a record of bonafide work carried out by me under the supervision of Prof. Malathy E.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: VIT, Vellore
Date: 02 / 06 / 2021

shaurya choudhary

Signature of the Candidate

CERTIFICATE

This is to certify that the project work entitled "QuickDrop: A File Sharing Hub" that is being submitted by **Shaurya Choudhary (18BCE2113)**, **SCOPE**, VIT, for *Internet and Web Programming (CSE3002)*, is a record of bonafide work done under my supervision during the period, 15. 02. 2021 to 01.06.2021. The contents of this Project Work, in full or in parts, have neither been taken from any other source nor have been submitted for any other CAL course. The Project Work fulfils the requirements and regulations of the Course and in my opinion meet the necessary standards for submission.

Place: VIT, Vellore
Date: 02 / 06 / 2021

shaurya choudhary
Signature of the Student

Signature of Faculty
Prof. Malathy E.

ACKNOWLEDGEMENTS

I would like to express my special thanks of gratitude to my teacher Prof. Malathy E who gave me the golden opportunity to do this wonderful project on Internet and Web Programming, which also helped me in doing a lot of Research and I came up with the project to understand the necessity and usefulness of the Web Applications for plethora of use cases. The project "*QuickDrop: A File Sharing Hub*" has helped me realize the potential and real-life applications of Internet and Web Programming in today's industry and I came to know about so many new things, I am really thankful to them.

Secondly, I would also like to thank my family who supported me throughout this project. I would also like to thank my friends who assisted me in finalizing this project within the limited time frame.

Shaurya Choudhary

Table of Contents

Table of Figures

1. ABSTRACT

Web Applications aren't just about websites. They're about experiences! File Sharing - It's an essential part of the digital world, especially during this pandemic, which allows family, friends and co-workers to share files on the go without any hassle. The internet has revolutionized the way people interact and share content with each other. The introduction of online file sharing and cloud services has made the traditional way of sharing files over CDs and Pen Drives with our friends and colleagues almost obsolete.

QuickDrop, is a file sharing Web Application project that comprises of all the needful features and provide a perfect solution for file management within a group. The project uses SQLite3 Database and is built using Django framework. QuickDrop is a lightweight solution and can be easily deployed by users for their personal use cases as well. The users can manage their profile and posts and share with the group without any hassle within seconds.

2. INTRODUCTION

To build a Web Application which will provide a File Sharing Platform which will mainly address the user segment of people in a group. The website will cut no edges in terms of functionality and will provide no-compromise full fledged hub for people to share files with their peers. We aim to develop a web app which will act as hub for users to easily share their files and access the files uploaded by others. All the files shared on the platform will contain the title and description, to give the exact idea about the post and make the experience user-friendly, users will also maintain and edit their Profile on website to keep their unique identity. This will be achieved through a secure user management system and we also aim to provide an admin dashboard where the admins can easily access, moderate and verify all the data flowing through the website and keep a check on the users.

2.1. SOFTWARE SPECIFICATIONS

Language/Tools used:

- → Client Side: HTML5, JavaScript, CSS3, Bootstrap5
- → Server side: Django-Python
- → Backend: SQLite3
- → IDE used: Visual Studio Code (VSC) and PyCharm for coding and designing the website.

2.1.1. Front-End/ Client-Side Programming

Hyper Text Markup Language (HTML) is the backbone of any website development process, without which a web page doesn't exist. Hypertext means that text has links, termed hyperlinks, embedded in it. A markup language indicates text can be turned into images, tables, links, and other representations. It is the HTML code that provides an overall framework of how the site will look. HTML was developed by Tim Berners-Lee. The latest version of HTML is called HTML5 and was published on October 28, 2014 by the W3 recommendation. This version contains new and efficient ways of handling elements such as video and audio files.

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is designed to enable the separation of presentation and content, including layout, colours, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate *.css* file, and reduce complexity and repetition in the structural content.

JavaScript often abbreviated as JS, is a high-level, interpreted programming language. Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web. JavaScript enables interactive web pages and thus is an essential part of web applications. The vast majority of websites use it, and all major web browsers have a dedicated JavaScript engine to execute it. JavaScript Code can use the Document Object Model (DOM), provided by the HTML standard, to manipulate a web page in response to events, like user input. Using a technique called AJAX, JavaScript code can also actively retrieve content from the web (independent of the original HTML page retrieval), and also react to server-side events as well, adding a truly dynamic nature to the web page experience.

Bootstrap5 is a free and open-source collection of CSS and JavaScript/jQuery code used for creating dynamic layout websites and web applications. Being a tool for creating front-end design, it consists of a series of HTML- and CSS-based design templates for different components of a website or application such as forms, buttons, navigation, modals, typography and other interface components with helpful JavaScript extensions.

2.1.2. Server-Side Programming

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

2.1.3. Backend/ Database

SQLite is a software library that provides a relational database management system. The lite in SQLite means lightweight in terms of setup, database administration, and required resources. SQLite has the following noticeable features: self-contained, serverless, zero-configuration, transactional. We have used SQLite3 database in our project.

2.2. HARDWARE SPECIFICATIONS

The project can be easily built with very minimal hardware requirements. Suggested configuration is used for the implementation of this project:

- Processor: i5 9300H
- RAM: 16GB DDR4
- Storage: 512 GB SSD
- GPU: GTX 1050
- Screen Resolution: At least 1024 x 760 required for proper and complete viewing of screens.
- Network Connection: A minimum speed of 64kbps is required.
- OS: Windows 10 21H1 64-Bit

3. EXISTING SYSTEM PROBLEMS

File Sharing is one of the most used services over the internet. As we dive deeper into the digital world, the need for a platform where users can securely share their files with their peers is more than ever. The users need a hub where they can share files without any restrictions and have access to those files at anytime and at any place.

There are tons of file sharing websites already out there. Since their introduction, sites like Google Drive, Dropbox, and OneDrive have attracted millions of users, many of whom have integrated these sites into their daily practices. As of completing this project, there are hundreds of services, with various technological affordances, supporting a wide range of interests and practices. While their key technological features are fairly consistent, the cultures that emerge around are varied. With all their benefits and advantages, they still cut a lot of edges and have issues which can be improved upon. Some of the problems are:

- Majority of the websites are paid.
- Free websites are not well built and have tons of ads.
- Very cluttered and less user-friendly design
- Lack of provision for sharing files within a group
- Upload size limit while sharing files
- Many formats of files are restricted
- Take very long time to upload files due to ton of traffic

4. PROPOSED SYSTEM DESIGN

This website is purely designed with the intension to cater the need of people who need a minimal website without any restrictions which will allow them to share files in an efficient and user-friendly manner without any hassle. The website offers a to-the-point service to allow users share files with their peers. The new users are asked to register themselves and get verification to avail all services available. A regular account holder simply needs to login to use the platform. The website will also have capability to maintain user account in a secured way holding various details using encryption.

Website Perspective:

The website will be self-contained, independent and accessible via any internet connection and web browser. Due to the nature and optimal build of the web app, it can be easily deployed and used for personal use as well.

User Interfaces:

The application will have a user-friendly interface. Following screens would be provided:

- A Main screen (Home Page) containing a brief introduction of the project.
- A Login screen for entering the username and password will be provided.
- A Signup page will allow new users to register themselves on the platform.
- There will also be an about page showing the basic details and goals of the website.
- User Profile page will show the details of the user.
- The Default Page will show all the files shared over the platform.
- My Posts Page for managing and Uploading files.

Website Functions:

This website will allow access only to authorized users with specific roles (administrator, user) depending upon user's role, he/she will be able to access only specific modules of the system.

A summary of the major functions that the website will perform are as follows:

- Login facility for enabling only authorized access to the system.
- User will be able to request/accept the selected entries from the list.
- Administrator will be able to add/modify/delete/update and alter data (i.e., product details) at the back-end as per the requirements.
- Administrator will be responsible for managing user account.

5. IMPLEMENTATION

The complete project is managed through my GitHub repository, which can be accessed from here: **https://github.com/shaurya-src/QuickDrop**.

Fig. 5.1. The RoadMap of Project

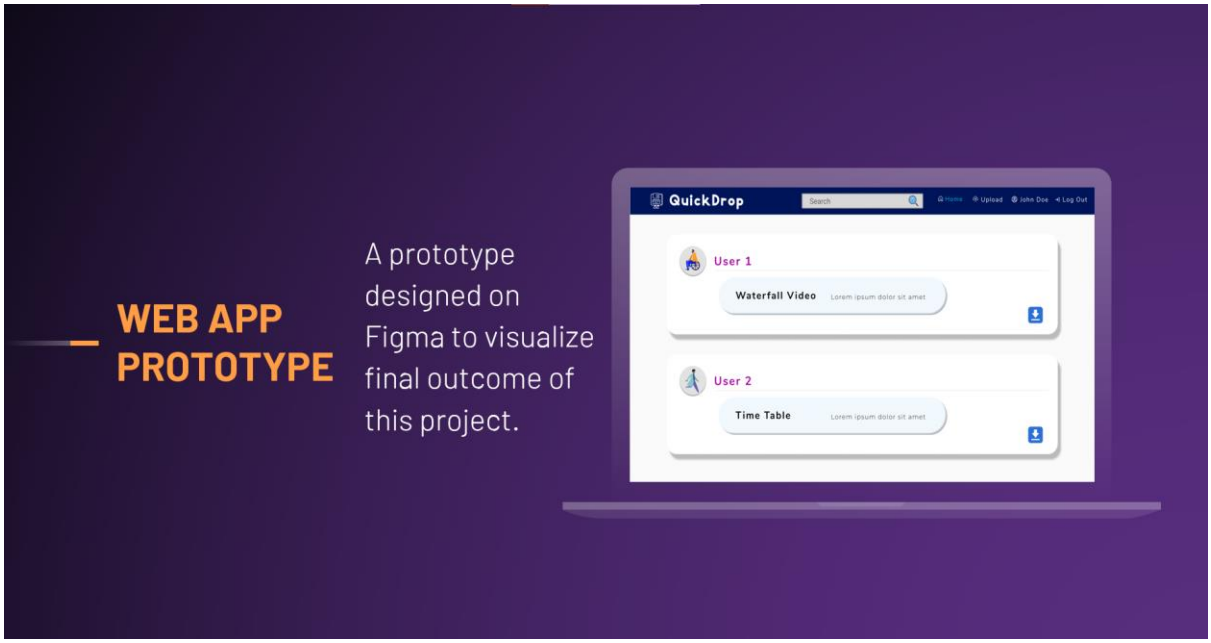

Fig. 5.2. Website Prototype using Figma

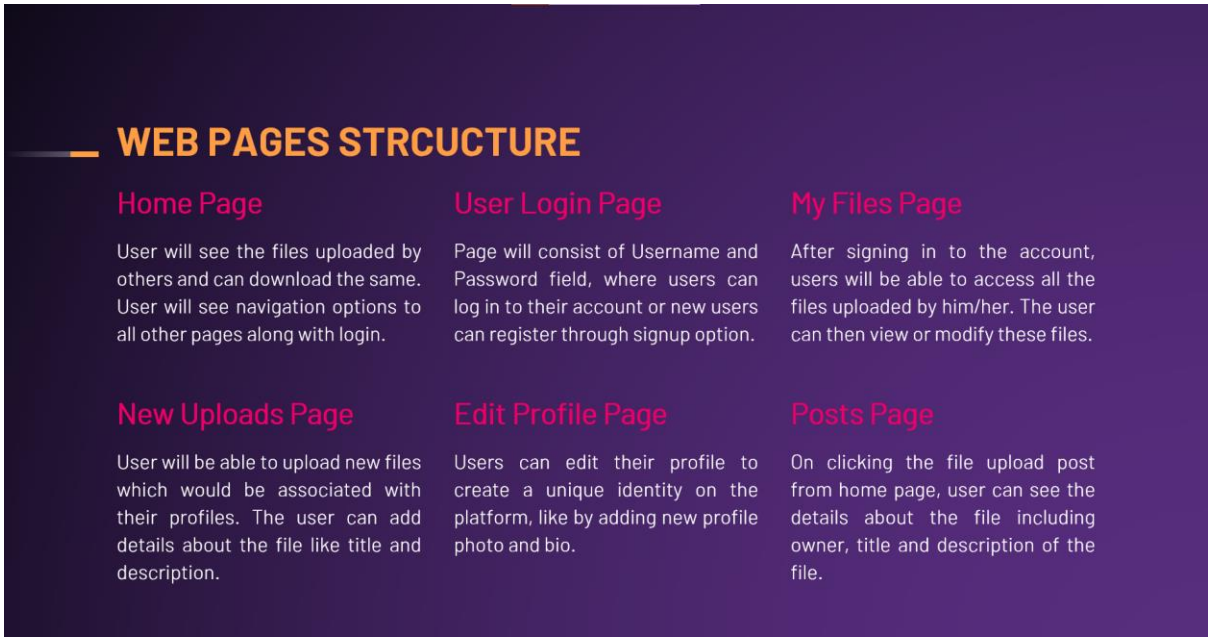

Fig. 5.3. Website Structure on User End

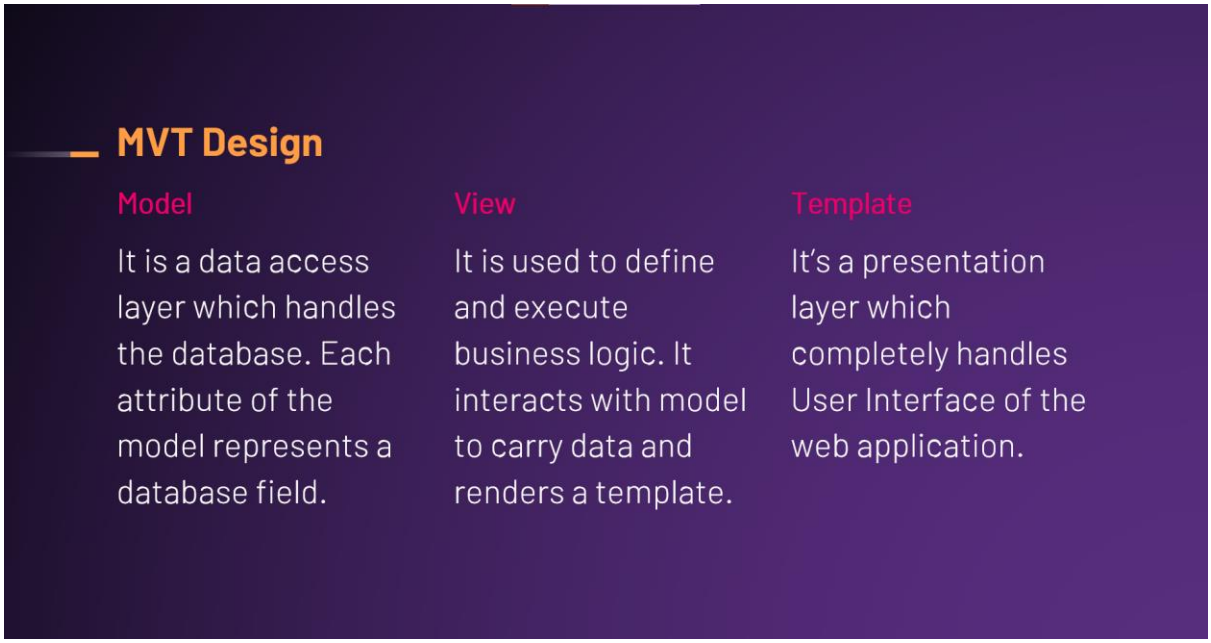

Fig. 5.4. System Design Approach for implementation

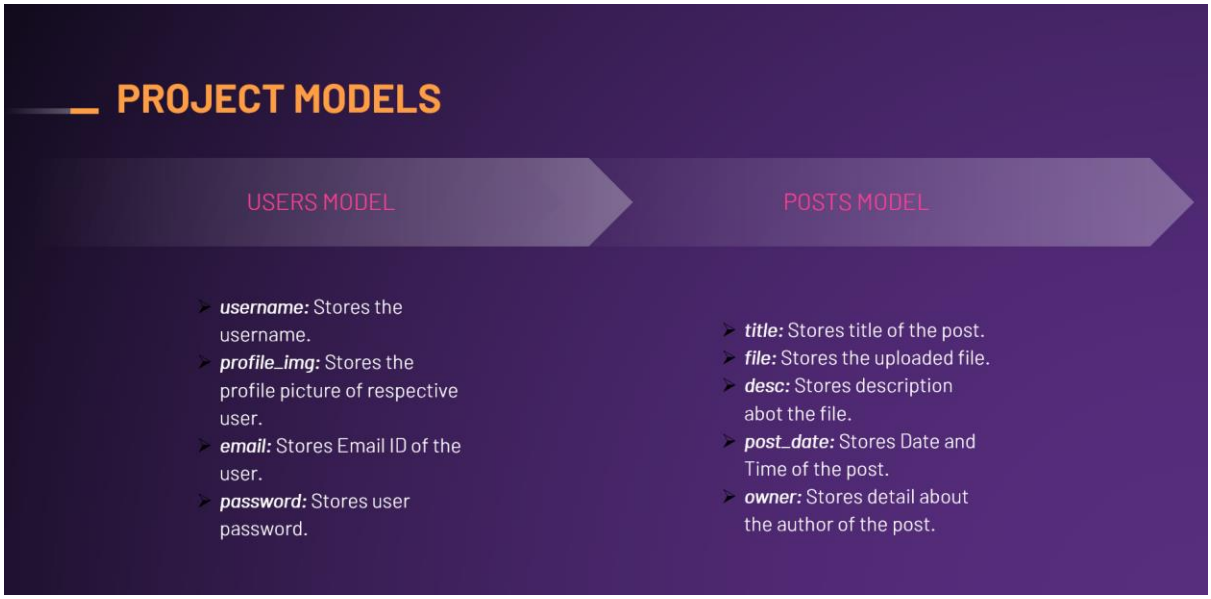

Fig. 5.5. Project Models for Database Design

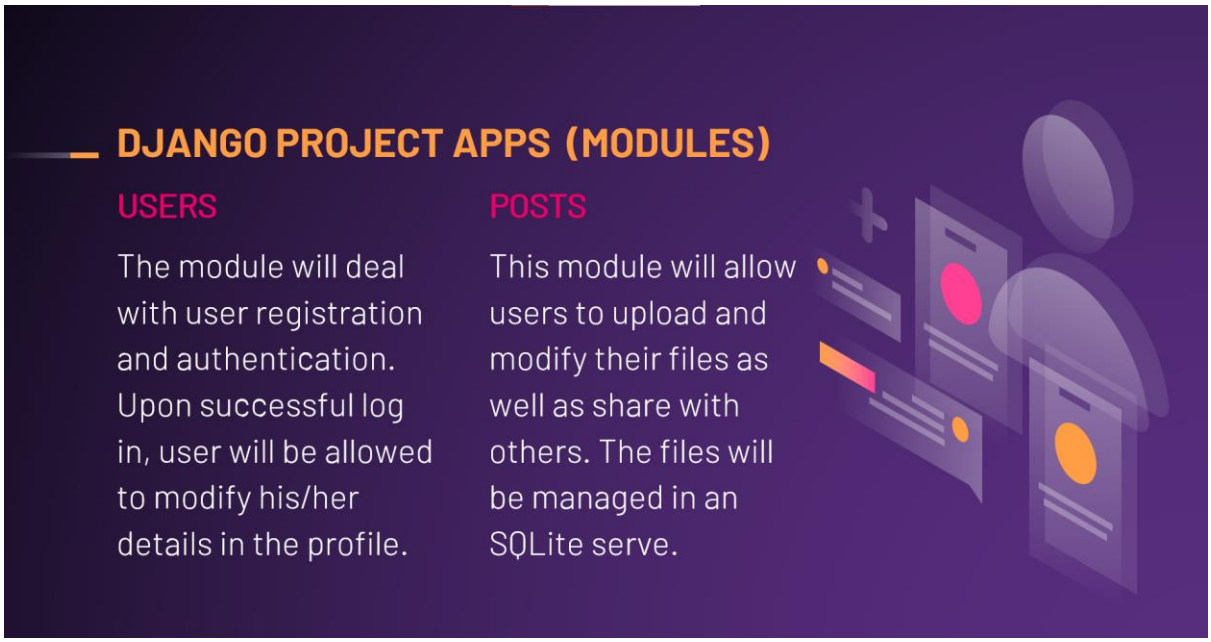

Fig. 5.6. Main project modules distribution

6. RESULTS

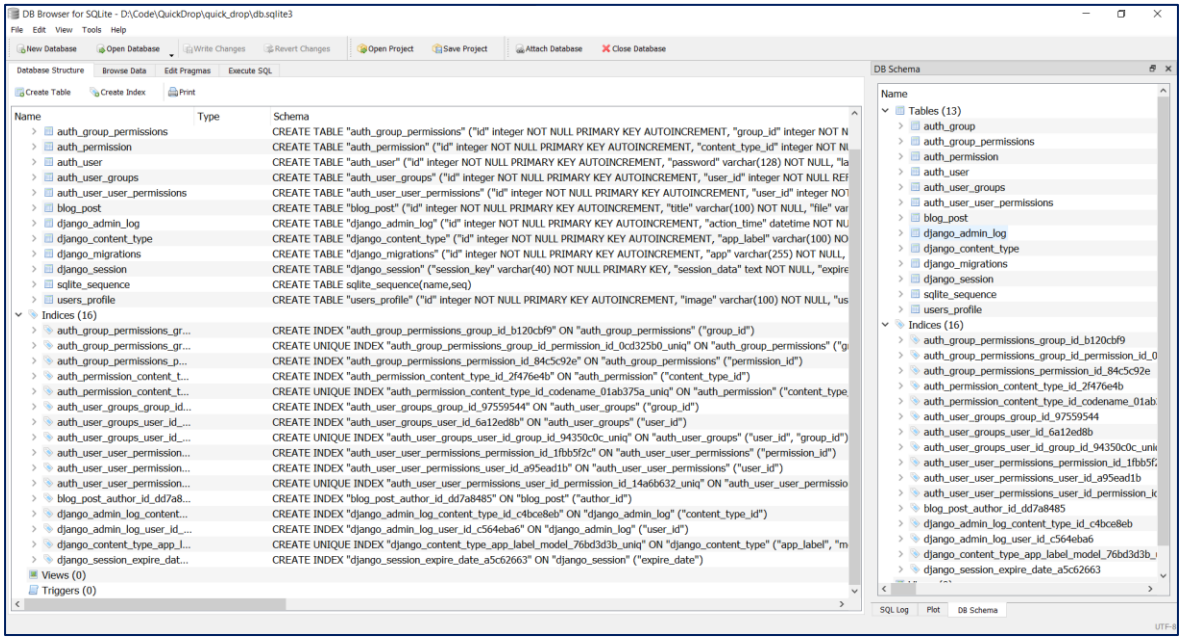

Fig. 6.1. Database Structure

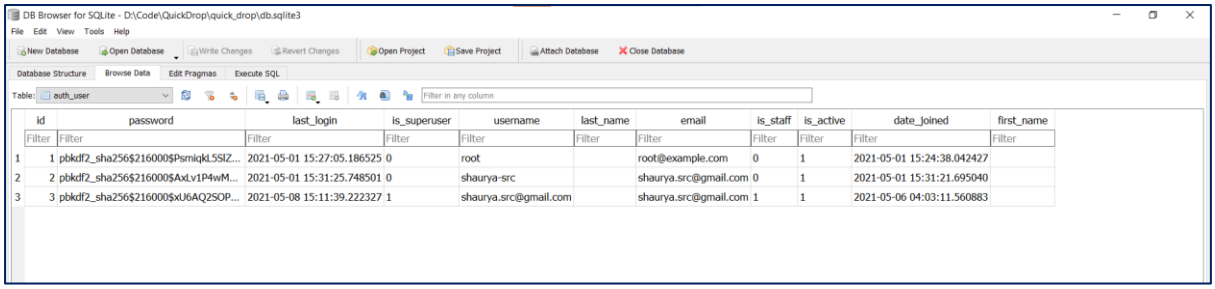

Fig. 6.2. Users Table

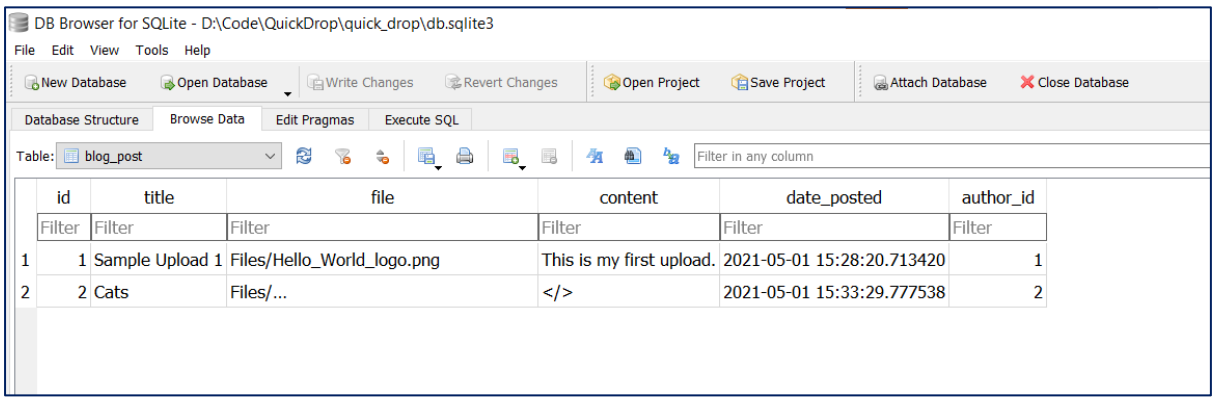

Fig. 6.3. User Posts Table

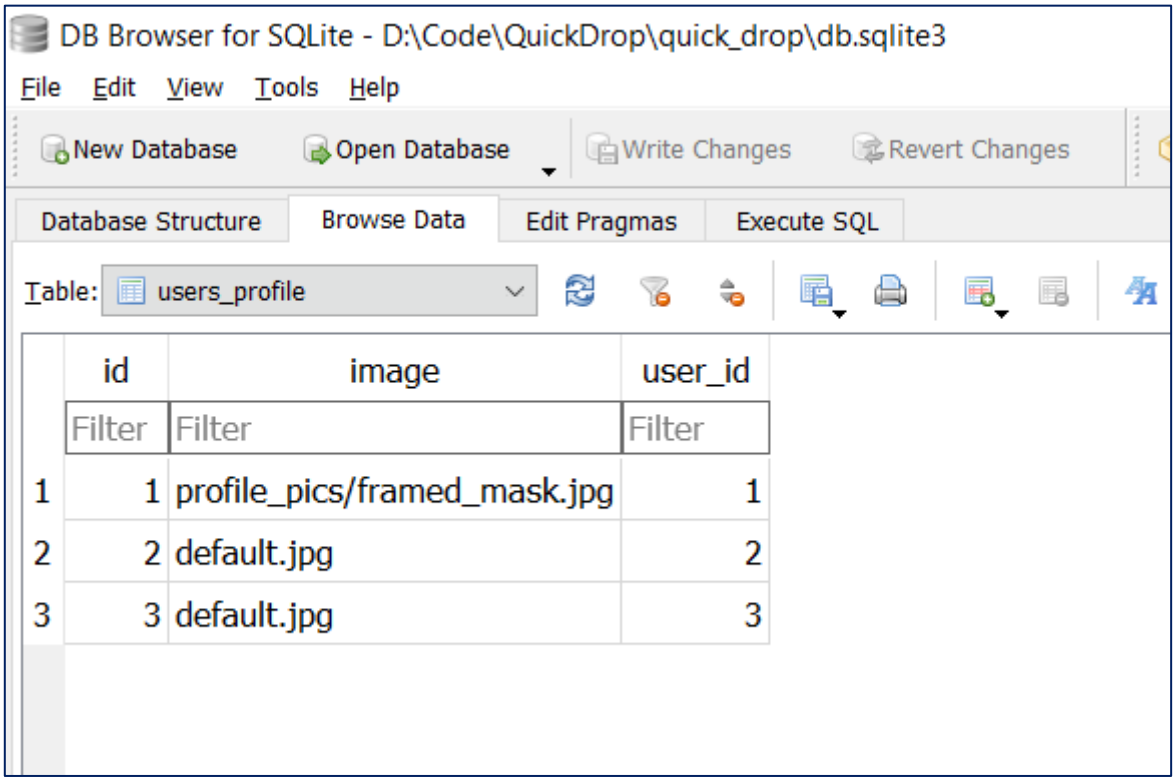

Fig. 6.4. User Profile Table

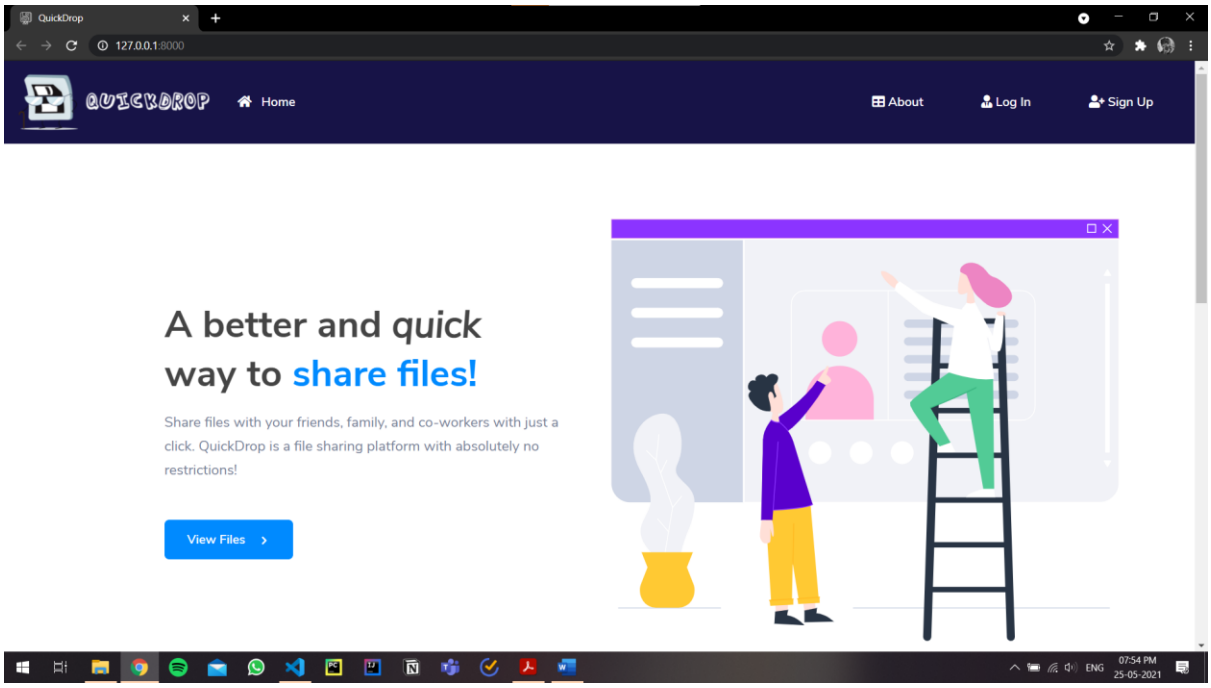

Fig. 6.5. Home Page

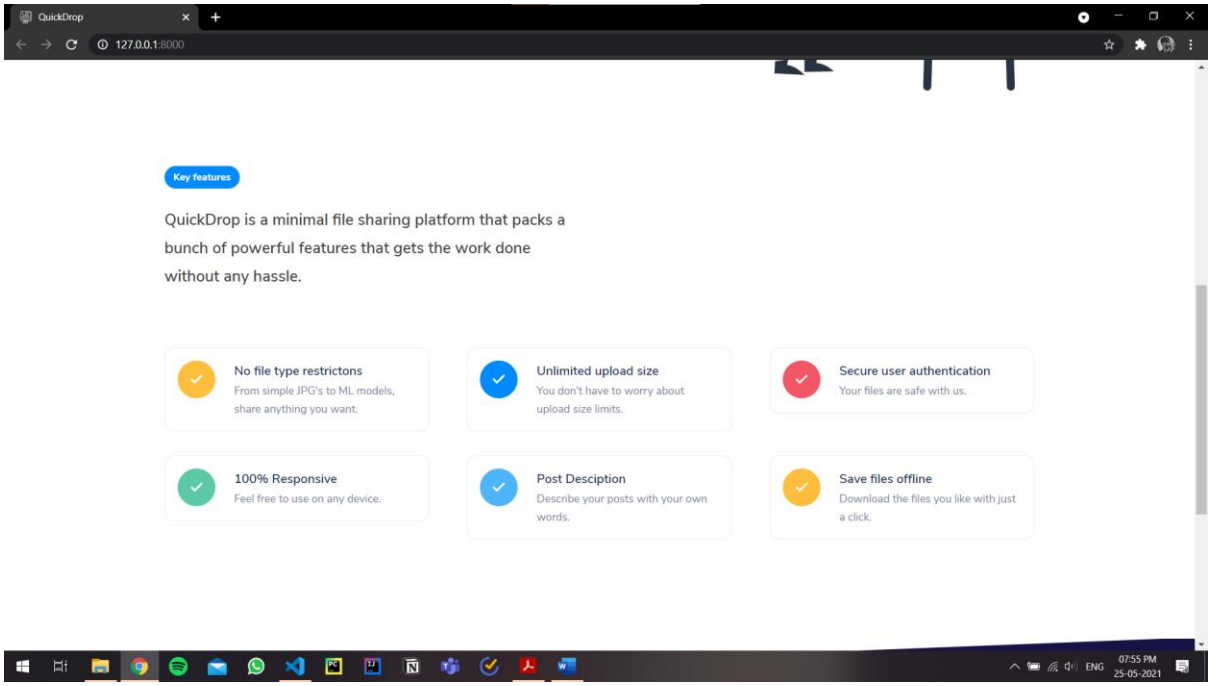

Fig. 6.6. Home Page (continue)

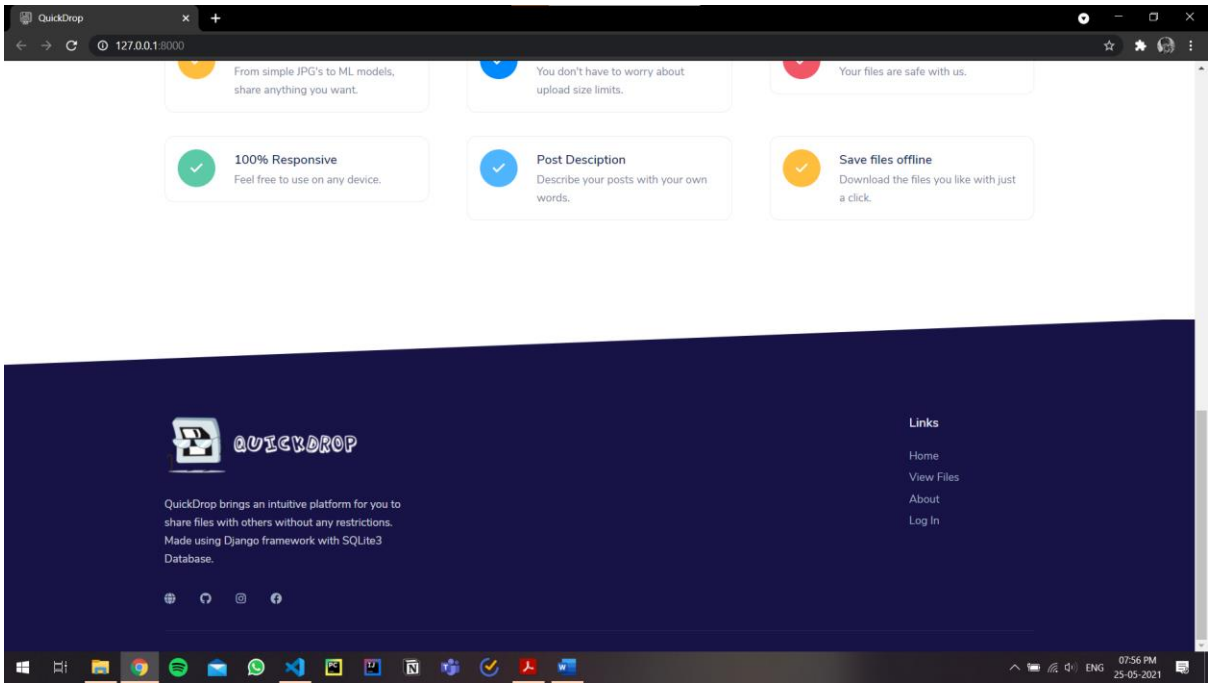

Fig. 6.7. Website Footer

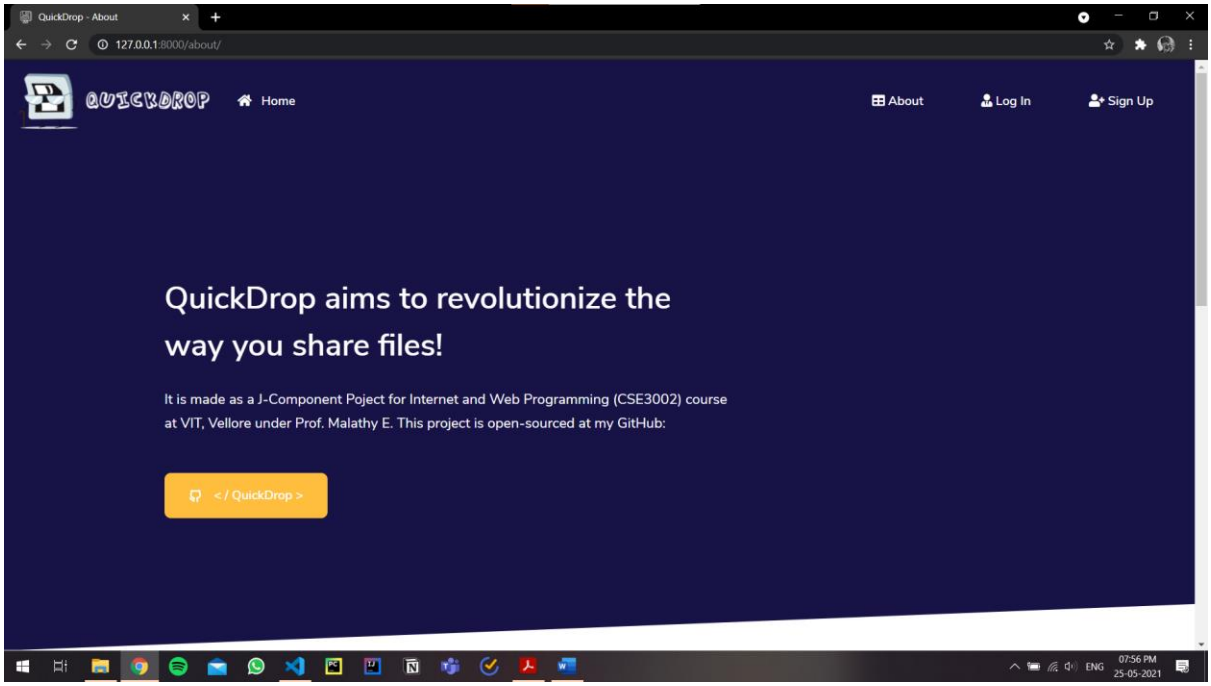

Fig. 6.8. About Page

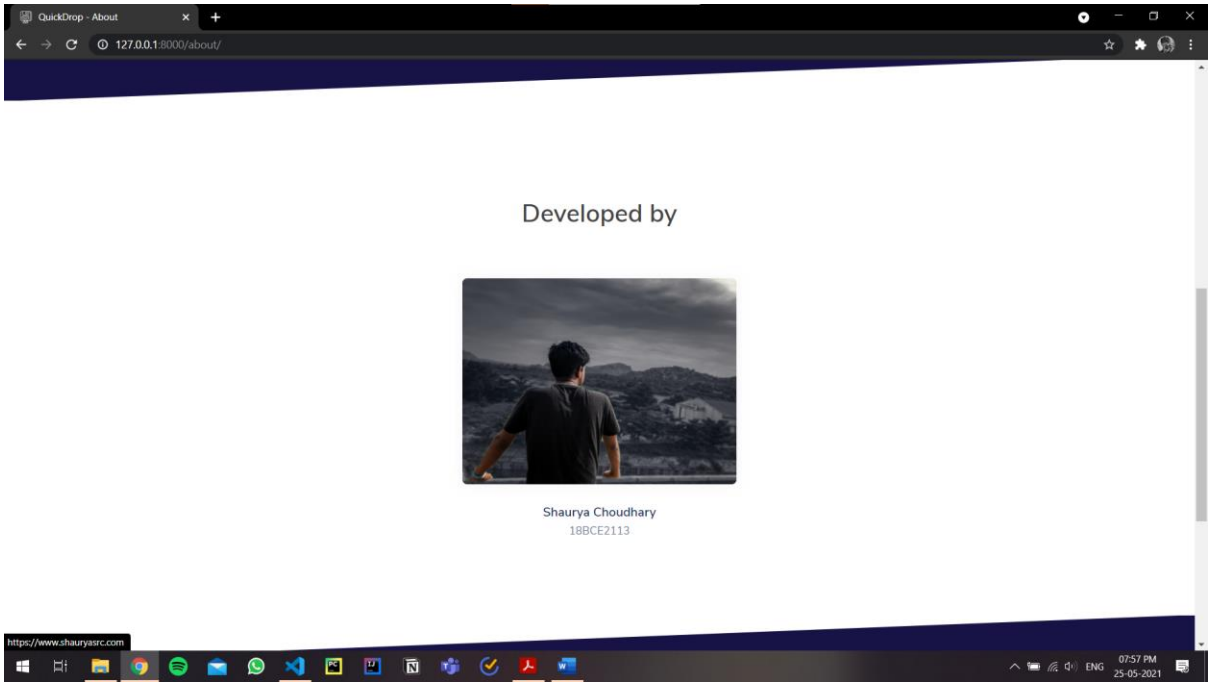

Fig. 6.9. About Page (continue)

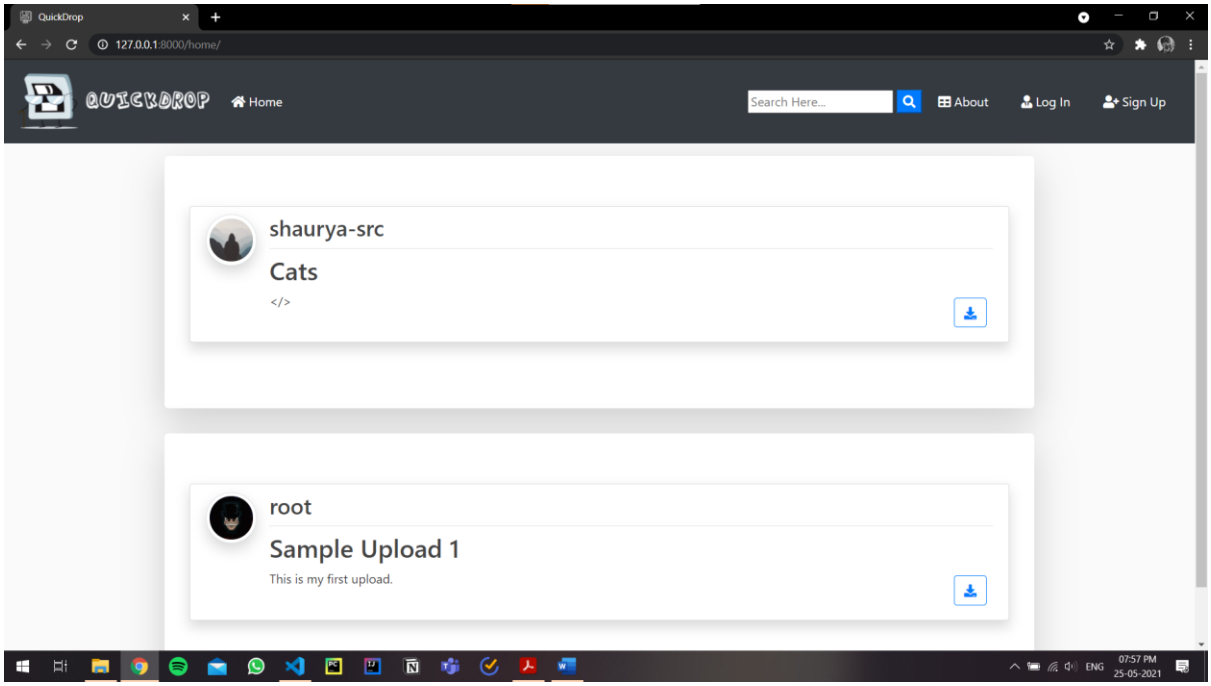

Fig. 6.10. Files Preview (No user Login)

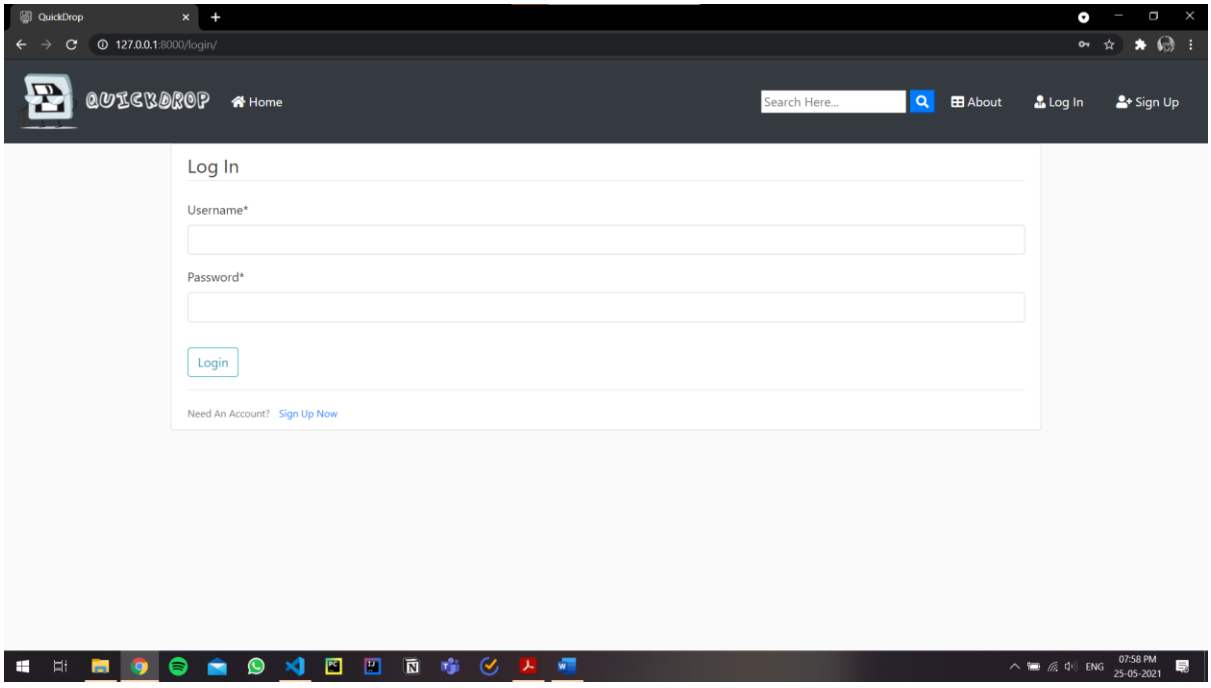

Fig. 6.11. User Login Page

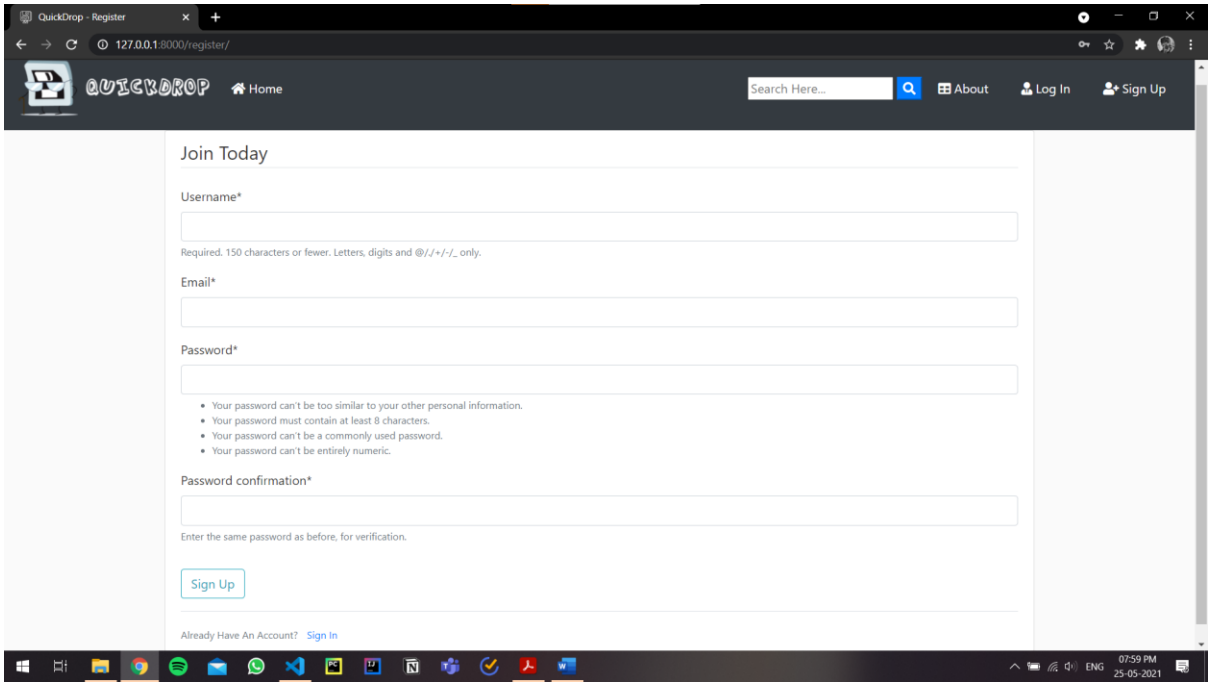

Fig. 6.12. Sign Up Page (for New User)

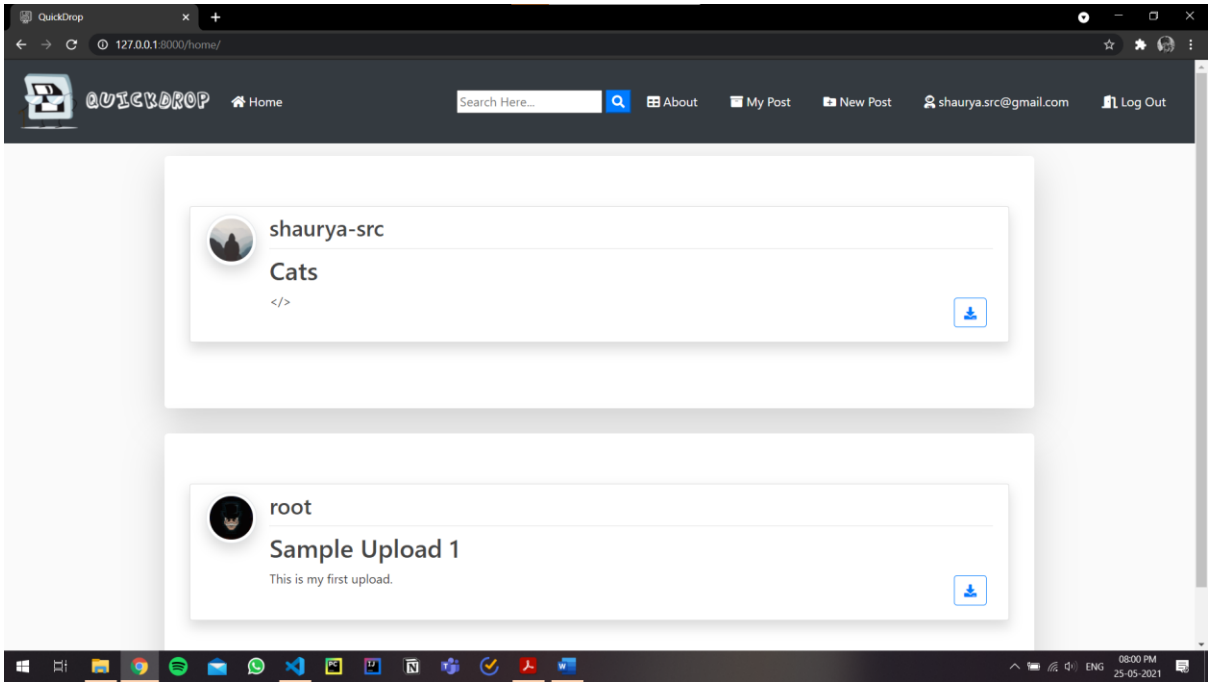

Fig. 6.13. Files Page (User Logged In)

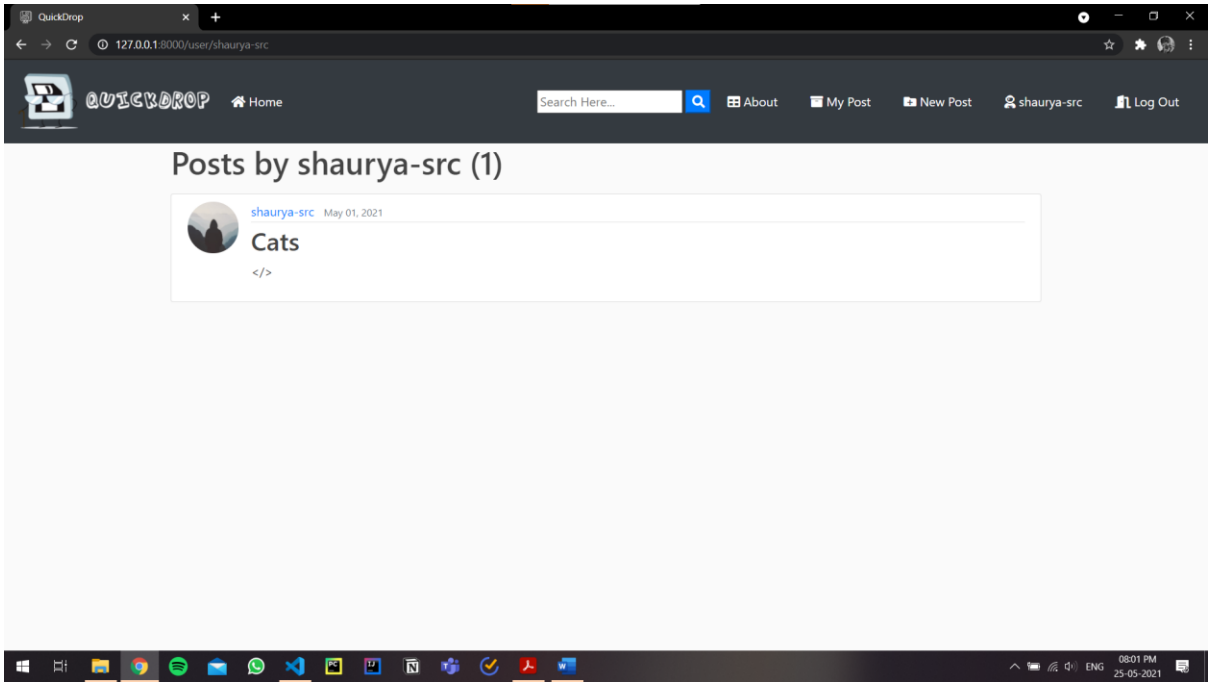

Fig. 6.14. My Posts Page

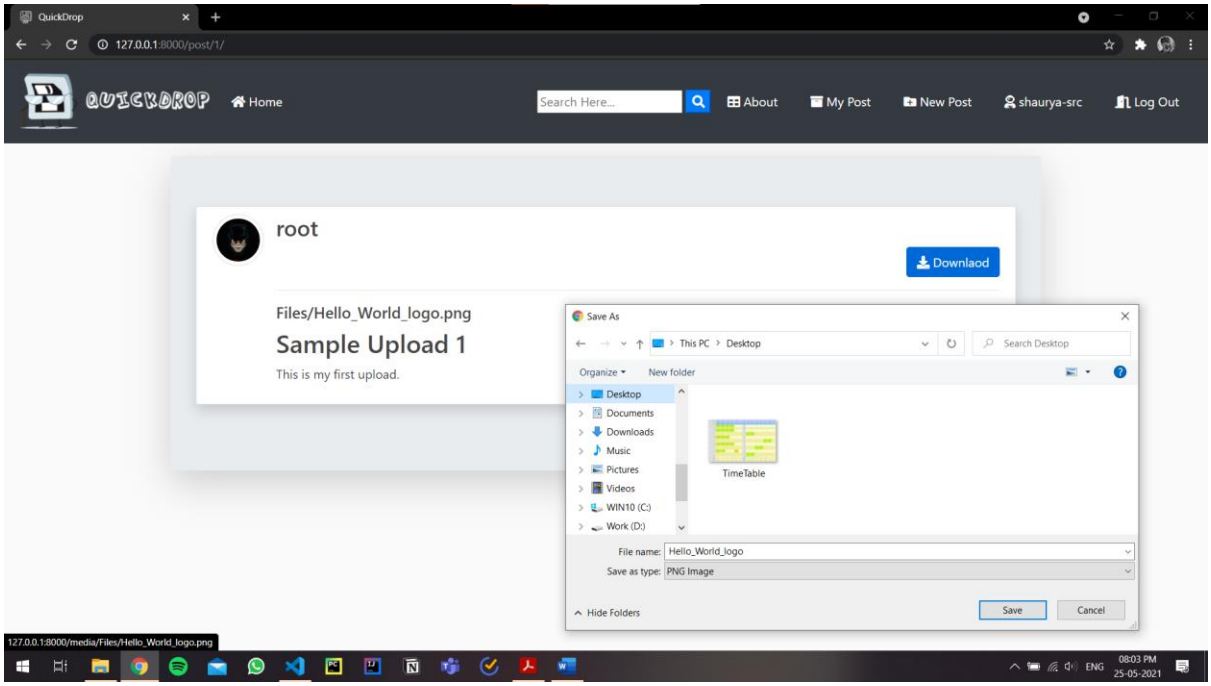

Fig. 6.15. File Preview/Download Page

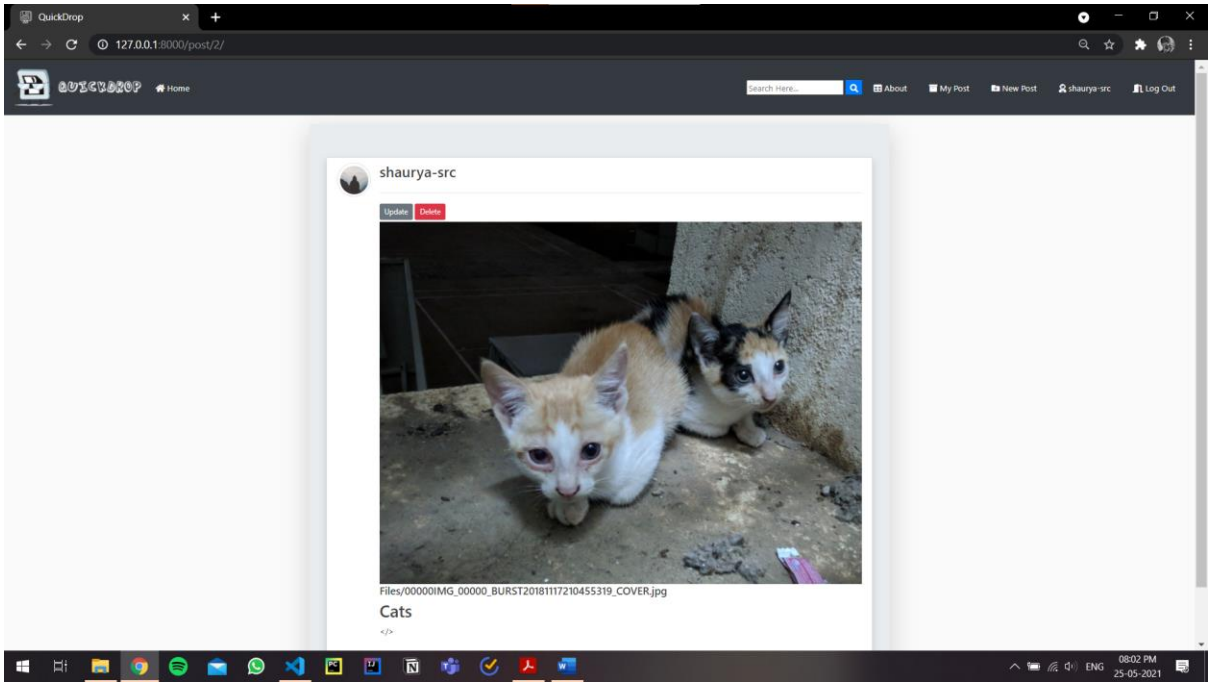

Fig. 6.16. Edit Posts Page

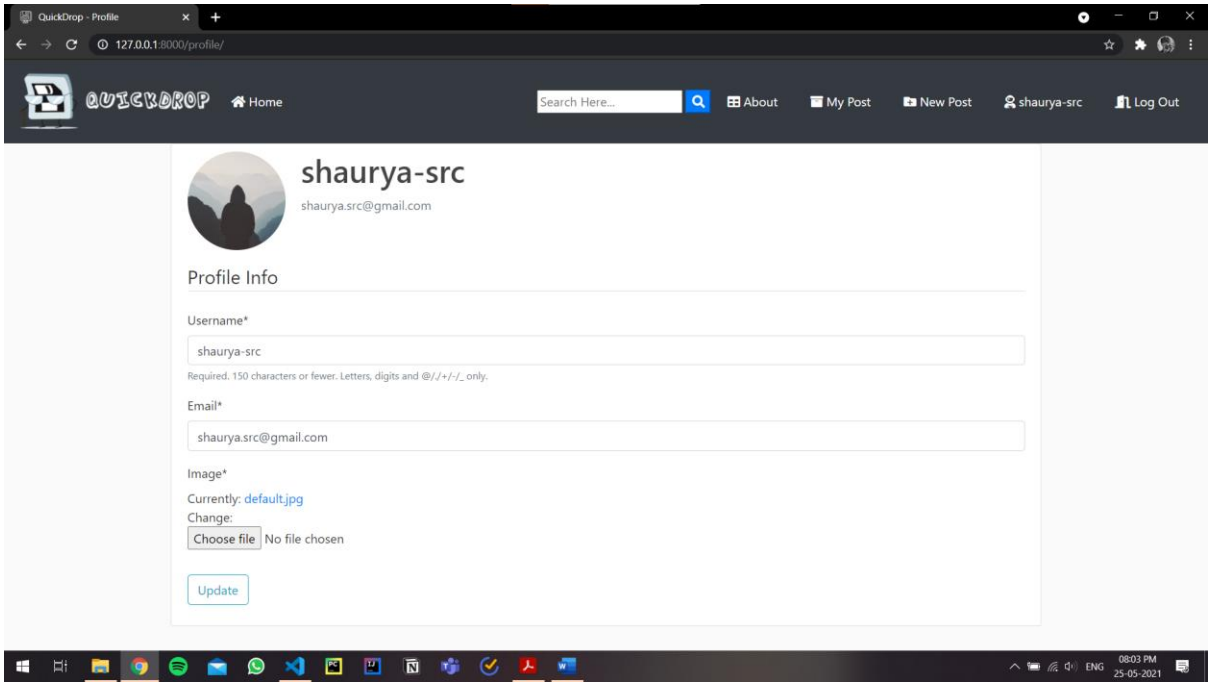

Fig. 6.17. User Profile Page

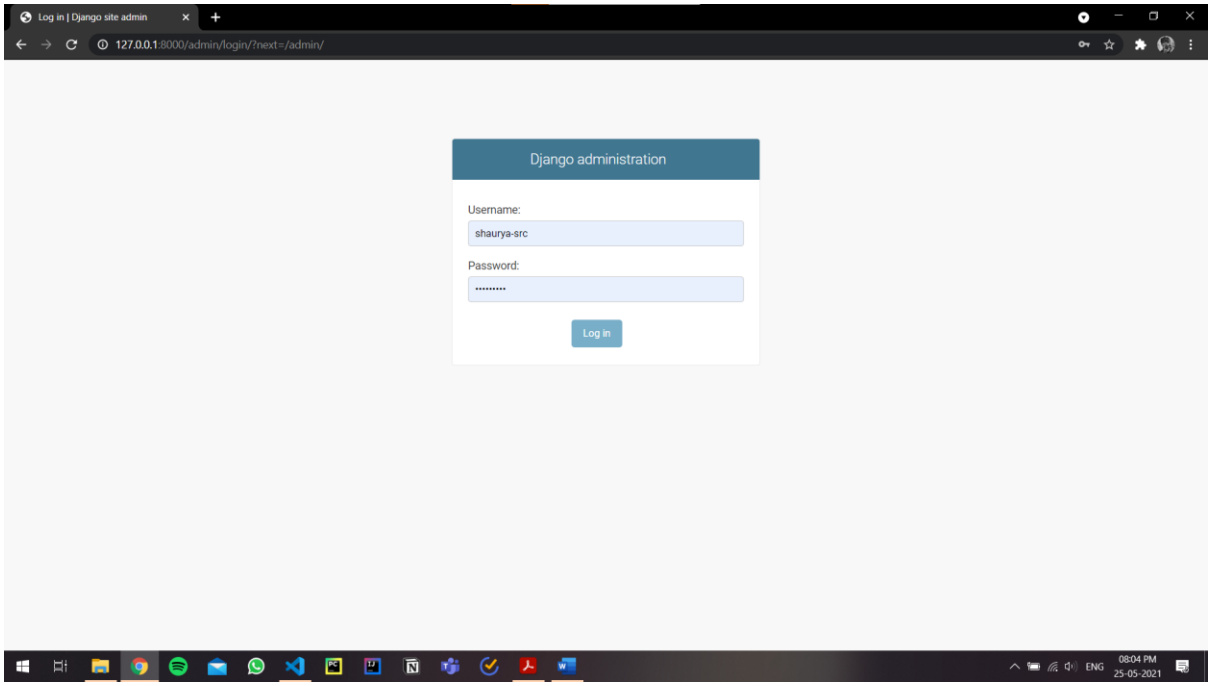

Fig. 6.18. Admin Login Page

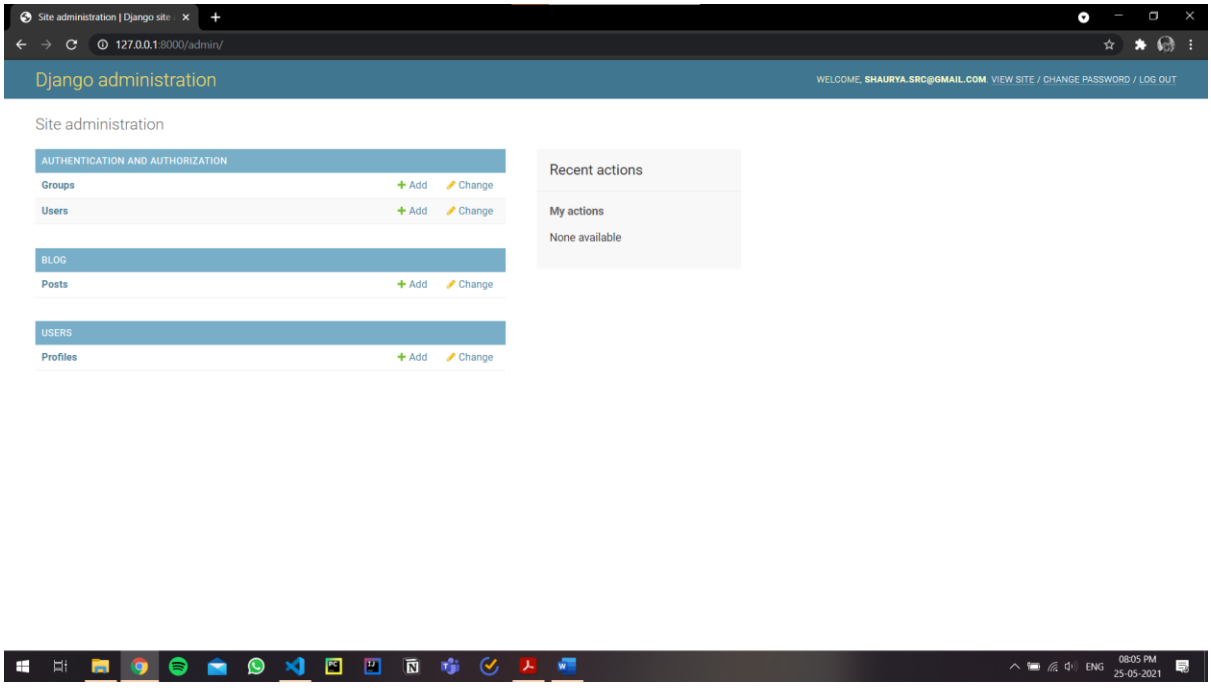

Fig. 6.19. Admin Dashboard

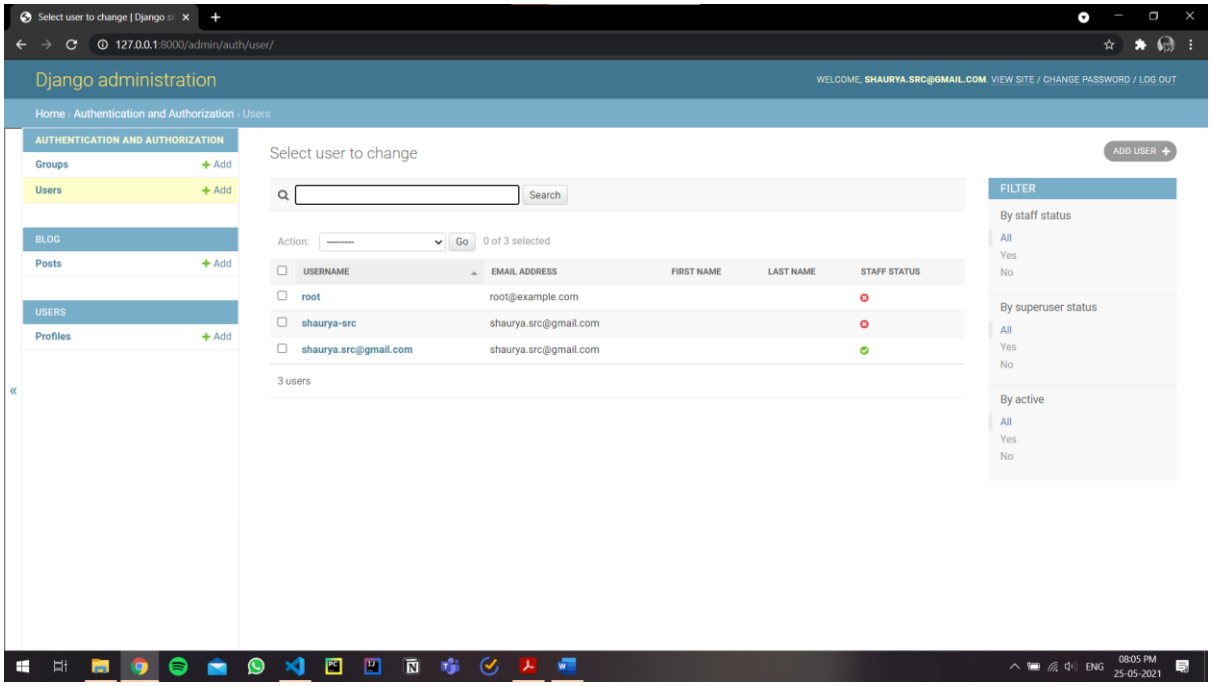

Fig. 6.20. Admin User Management

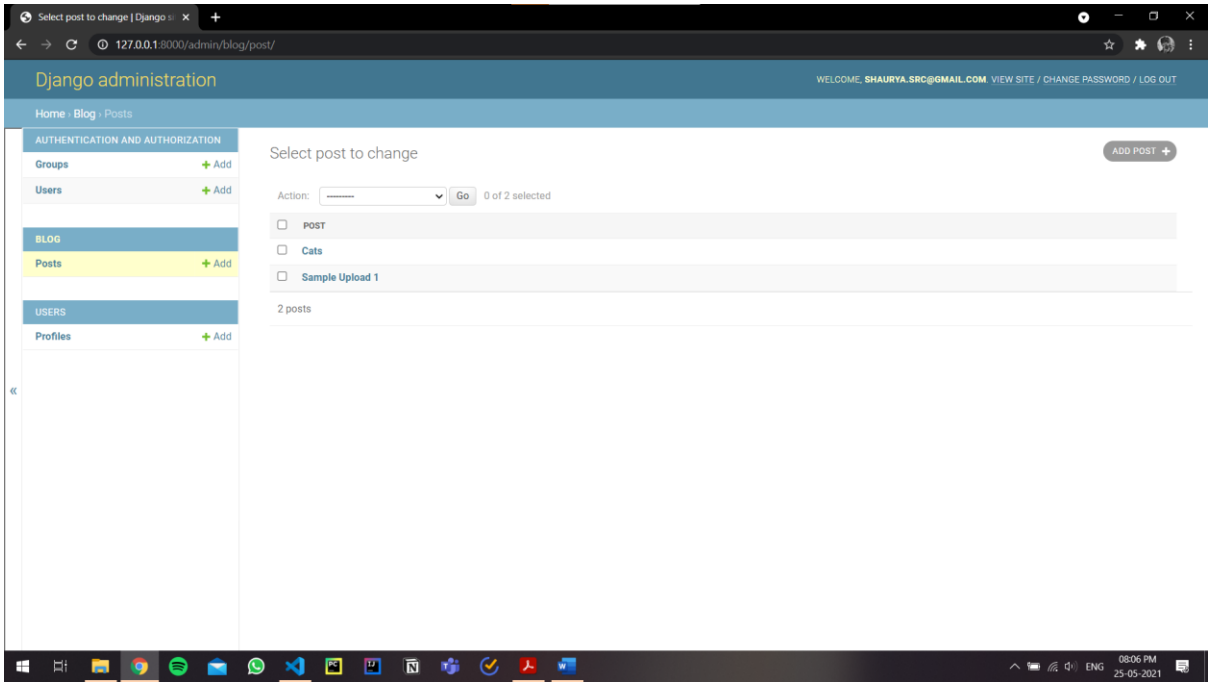

Fig. 6.21. Admin Posts Overview

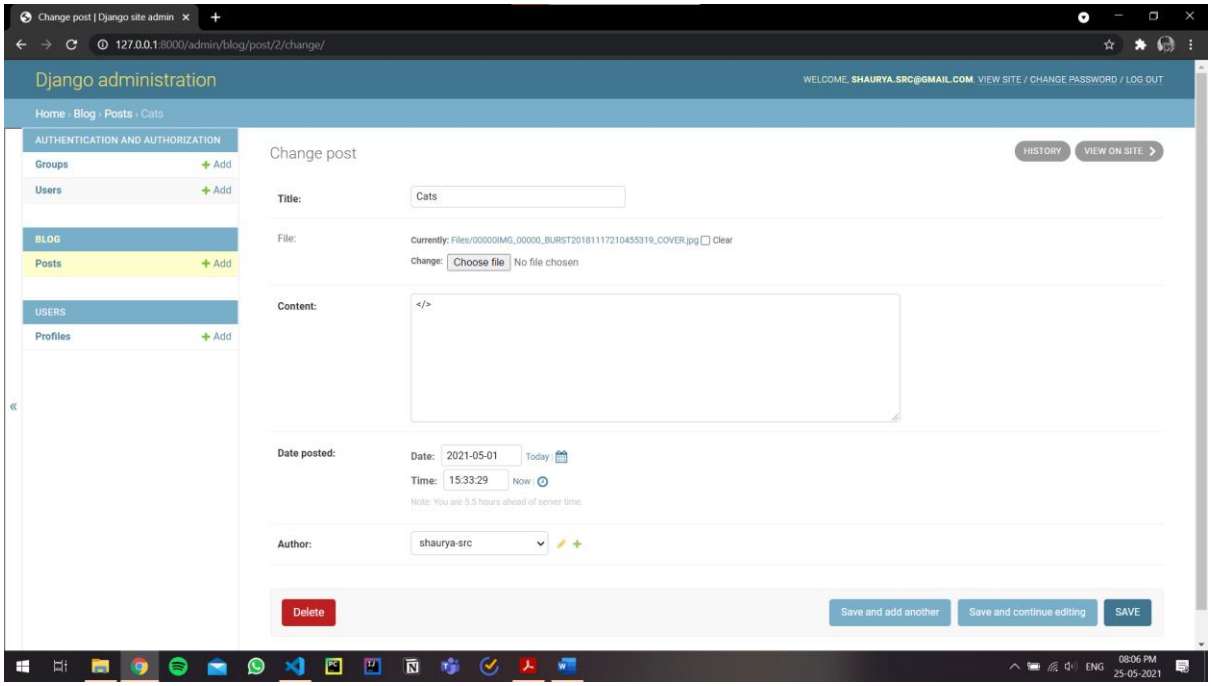

Fig. 6.22. Admin Posts Management

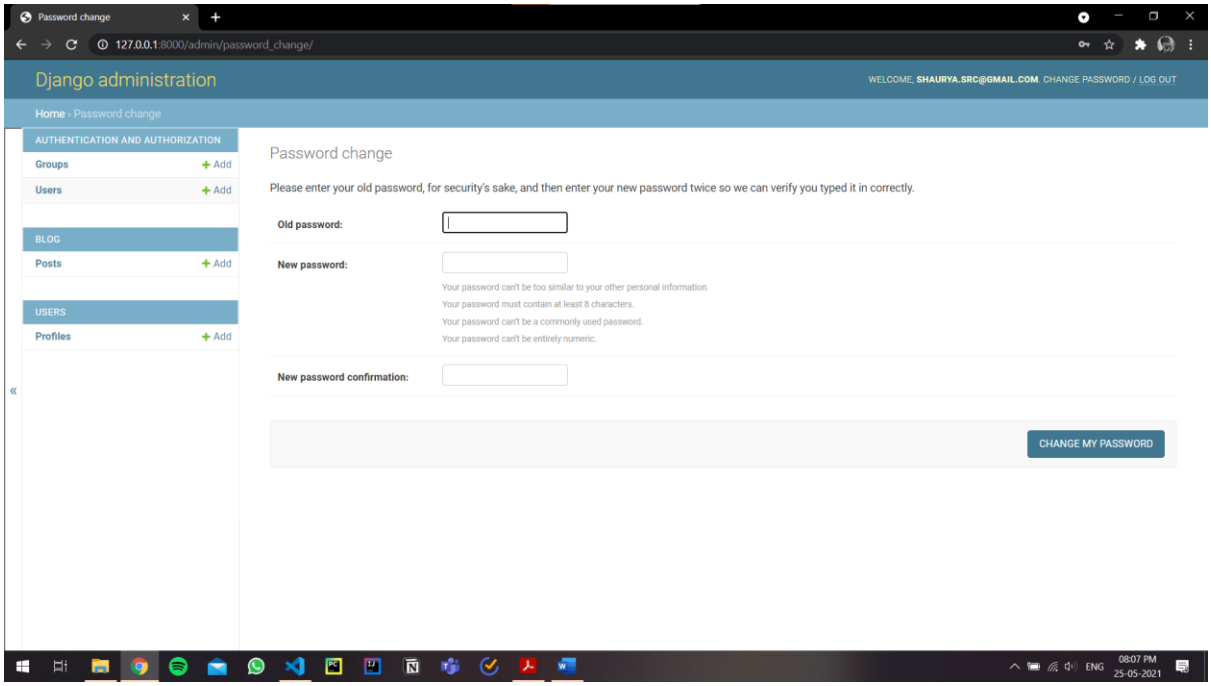

Fig. 6.23. Update Admin Credentials

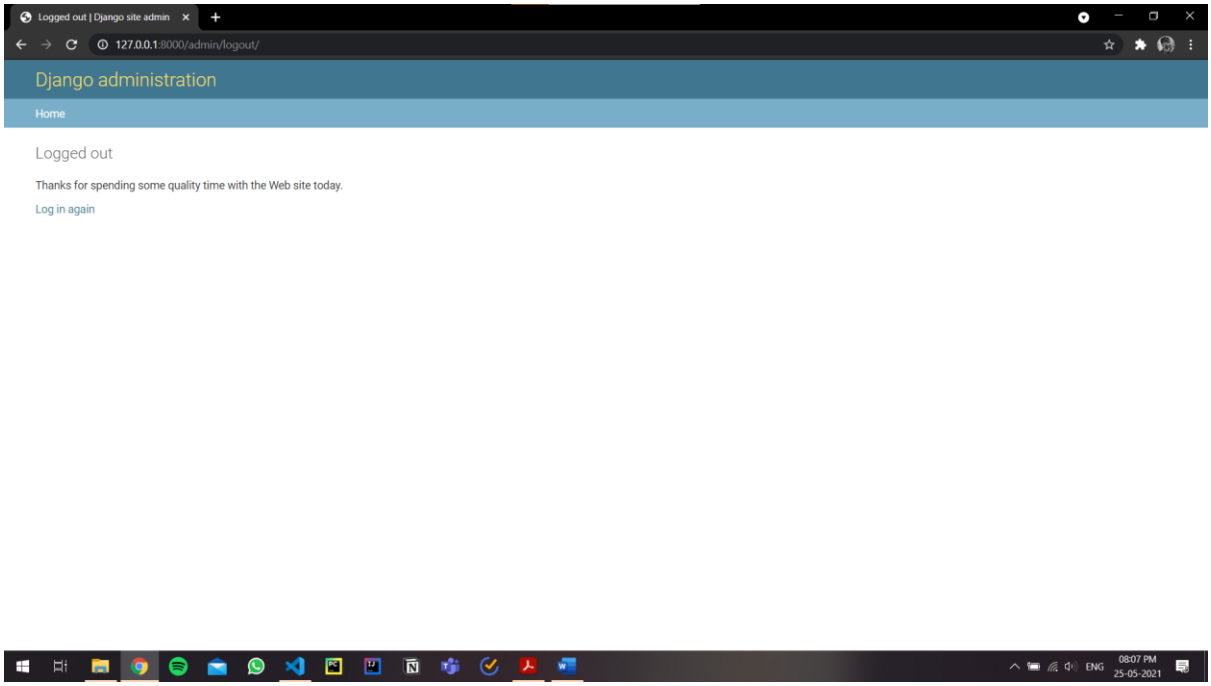

Fig. 6.24. Admin Logged Out Page

7. CONCLUSION AND FUTURE WORK

While developing the system, a conscious effort has been made to create and develop a web application, making use of available tools, techniques and resources – that would generate a proper system for a novel file sharing platform. The main focus was on making it as user-friendly and minimal without making any compromises. As such one may hope that the system will be acceptable to any user and will adequately meet his/her needs. As in case of any system development process where there are a number of short comings, there have been some shortcomings in the development of this system also.

The project has been developed with Django and SQLite3 database keeping in mind the specification of the system. Overall, the project teaches us the essential skills like:

- Using system analysis and design techniques to develop a project
- Understanding the database handling
- Understanding and applying the logic required for file sharing platform

There are some of the areas of improvement which couldn't be implemented due to time constraints. One such feature was comment section where members can comment on the shared posts and interact with other users through this website.

Future scope of this project will incorporate the following features:

- Adding features of User based Communities/Groups
- Live Comment Section for users
- Option to upload multiple files in a single post
- Incorporate ML models to detect malicious uploads

8. REFERENCES

[1] Potential Problems: https://source.raysync.io/news/how-to-solve-the-problem-of-file-sharing

[2] Django Help: https://docs.djangoproject.com/en/3.2/topics/auth/default/

[3] Bootstrap Ref: https://getbootstrap.com/docs/4.3/getting-started/introduction/

[4] SQLite DB Viewer: https://sqlitebrowser.org/

[5] Best Practices: https://www.laac.dev/blog/five-advanced-django-tips/

[6] Icons: https://fontawesome.com/icons?d=gallery&p=2

[7] File Sharing Security: https://www.cleo.com/blog/knowledge-base-secure-file-exchange

9. CODE LINK

GitHub: https://github.com/shaurya-src/QuickDrop

10. APPENDIX

QuickDrop Project

settings.py

```
from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'w4bx=t%u6e6xnk6lezo&zrpyvzs=^-gi+%#62q6dli^k+by1s6'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog.apps.BlogConfig',
    'users.apps.UsersConfig',
    'crispy_forms',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
```

```
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'quick_drop.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'quick_drop.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
```

```
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidat
or',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValida
tor',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT= os.path.join(BASE_DIR, 'static'),

MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'

CRISPY_TEMPLATE_PACK = 'bootstrap4'

LOGIN_REDIRECT_URL = 'blog-home'
LOGIN_URL = 'login'
```

urls.py

```
"""quick_drop URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.1/topics/http/urls/
Examples:
Function views
```

```
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
from users import views as user_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('login/', auth_views.LoginView.as_view(template_name='users/login.html'), name='login'),
    path('register/', user_views.register, name='register'),
    path('profile/', user_views.profile, name='profile'),
    path('logout/', auth_views.LogoutView.as_view(template_name='users/logout.html'), name='logout'),
    path('', include('blog.urls')),
]


if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Blog App

models.py

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User
from django.urls import reverse
import os

class Post(models.Model):
    title = models.CharField(max_length=100)
    file = models.FileField(null=True,blank=True,upload_to='Files')
```

```
    content = models.TextField()
    date_posted = models.DateTimeField(default=timezone.now)
    author = models.ForeignKey(User, on_delete=models.CASCADE)

    def __str__(self):
        return self.title

    def extension(self):
        name, extension = os.path.splitext(self.file.name)
        return extension

    def get_absolute_url(self):
        return reverse('post-detail', kwargs={'pk': self.pk})
```

urls.py

```
from django.urls import path
from .views import (
    PostListView,
    PostDetailView,
    PostCreateView,
    PostUpdateView,
    PostDeleteView,
    UserPostListView
)
from . import views

urlpatterns = [
    path('', views.index, name='blog-index'),
    path('home/', PostListView.as_view(), name='blog-home'),
    path('user/<str:username>', UserPostListView.as_view(), name='user-
posts'),
    path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
    path('post/new/', PostCreateView.as_view(), name='post-create'),
    path('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-
update'),
    path('post/<int:pk>/delete/', PostDeleteView.as_view(), name='post-
delete'),
    path('media/Files/<int:pk>',PostDeleteView.as_view(),name='post-delete' ),
    path('search/',views.search,name='search' ),
    path('about/', views.about, name='blog-about'),
]
```

views.py

```
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponse
from django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin
from django.contrib.auth.models import User
from django.views.generic import (
    ListView,
    DetailView,
    CreateView,
    UpdateView,
    DeleteView
)
from .models import Post
import operator
from django.urls import reverse_lazy
from django.contrib.staticfiles.views import serve

from django.db.models import Q


def home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html', context)

def search(request):
    template='blog/home.html'

    query=request.GET.get('q')

    result=Post.objects.filter(Q(title__icontains=query) | Q(author__username_
_icontains=query) | Q(content__icontains=query))
    paginate_by=2
    context={ 'posts':result }
    return render(request,template,context)



def getfile(request):
   return serve(request, 'File')


class PostListView(ListView):
    model = Post
    template_name = 'blog/home.html'  # <app>/<model>_<viewtype>.html
    context_object_name = 'posts'
```

```
    ordering = ['-date_posted']
    paginate_by = 2


class UserPostListView(ListView):
    model = Post
    template_name = 'blog/user_posts.html'  # <app>/<model>_<viewtype>.html
    context_object_name = 'posts'
    paginate_by = 2

    def get_queryset(self):
        user = get_object_or_404(User, username=self.kwargs.get('username'))
        return Post.objects.filter(author=user).order_by('-date_posted')


class PostDetailView(DetailView):
    model = Post
    template_name = 'blog/post_detail.html'


class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    template_name = 'blog/post_form.html'
    fields = ['title', 'content', 'file']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)


class PostUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
    model = Post
    template_name = 'blog/post_form.html'
    fields = ['title', 'content', 'file']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False


class PostDeleteView(LoginRequiredMixin, UserPassesTestMixin, DeleteView):
    model = Post
    success_url = '/'
```

```
    template_name = 'blog/post_confirm_delete.html'

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False


def about(request):
    return render(request, 'blog/about.html', {'title': 'About'})

def index(request):
    return render(request, 'blog/index.html')
```

Users App

forms.py

```
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from .models import Profile


class UserRegisterForm(UserCreationForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']


class UserUpdateForm(forms.ModelForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email']


class ProfileUpdateForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ['image']
```

models.py

```
from django.db import models
from django.contrib.auth.models import User
from PIL import Image


class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    image = models.ImageField(default='default.jpg', upload_to='profile_pics')

    def __str__(self):
        return f'{self.user.username} Profile'

    def save(self, *args, **kwargs):
        super(Profile, self).save(*args, **kwargs)

        img = Image.open(self.image.path)

        if img.height > 300 or img.width > 300:
            output_size = (300, 300)
            img.thumbnail(output_size)
            img.save(self.image.path)
```

signals.py

```
from django.db.models.signals import post_save
from django.contrib.auth.models import User
from django.dispatch import receiver
from .models import Profile


@receiver(post_save, sender=User)
def create_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)


@receiver(post_save, sender=User)
def save_profile(sender, instance,created, **kwargs):
    instance.profile.save()
```

views.py

```
from django.shortcuts import render, redirect
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from .forms import UserRegisterForm, UserUpdateForm, ProfileUpdateForm


def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            messages.success(request, f'Your account has been created! You are
 now able to log in')
            return redirect('login')
    else:
        form = UserRegisterForm()
    return render(request, 'users/register.html', {'form': form, 'title': 'Reg
ister'})
@login_required
def profile(request):
    if request.method == 'POST':
        u_form = UserUpdateForm(request.POST, instance=request.user)
        p_form = ProfileUpdateForm(request.POST,
                                   request.FILES,
                                   instance=request.user.profile)
        if u_form.is_valid() and p_form.is_valid():
            u_form.save()
            p_form.save()
            messages.success(request, f'Your account has been updated!')
            return redirect('profile')
    else:
        u_form = UserUpdateForm(instance=request.user)
        p_form = ProfileUpdateForm(instance=request.user.profile)

    context = {
        'u_form': u_form,
        'p_form': p_form,
        'title' : 'Profile'
    }

    return render(request, 'users/profile.html', context)
```