



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR INSTITUTE OF TECHNOLOGY

A MINI PROJECT REPORT ON

“Create an online Social Website using REST APIs”

Submitted for V Semester Mini Project by

D SAI KUMAR REDDY (1CR19CS035)

DARSHAN R (1CR19CS035)

DUSI UJWAL (1CR19CS051)

Under the Guidance of,

Dr. Sugato Chakrabarty

Professor, Dept. of CSE

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI, BANGALORE-560037

ABSTRACT

The digital age has revolutionized how information is shared among human beings. The Internet initially provided a means for obtaining information and then evolved to allow the exchange of information between humans and Web sites. The enormous impact of these changes on health care has shifted the way physicians provide care and how patients elect for and receive care. Social media applications allow for immediate exchange of ideas between large populations, which presents many opportunities and challenges for practicing physicians. Providers must be cognizant of patient confidentiality, their own online reputation, and risk management when using social media. The future is widely unknown with opportunities for marketing, networking, and research to evolve in the coming decades.

The main theme of the project is to build a social media website where the user can do the following things:

- Register if new user
- Login
- View the feeds
- Add posts
- Share the posts
- Comment on posts
- Like the post
- Share photos
- Share Videos
- Follow and Unfollow
- Messaging

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

We would like to extend our sincere esteems towards our guide, **Dr.SUGATO CHAKRABARTY** for the support, guidance and encouragement, she provided during the BE Project. This work would have not been possible without her valuable time, patience and motivation. We thank her for making our stint thoroughly pleasant and enriching. We are deeply indebted to Dr. Shreekanth M. Prabhu (Head of Department of CSE) and the entire team in the Computer Department. They supported us with scientific guidance, advice and encouragement, they were always helpful and enthusiastic and this inspired us in our work. We take the privilege to express our sincere thanks to **Dr. Sanjay Jain**, our Principal for providing the encouragement and much support throughout our work.

DADIREDDY SAI KUMAR REDDY
DARSHAN R
DUSI UJWAL

TABLE OF CONTENTS

	Page No.
Abstract	ii
Acknowledgement	iii
Table of contents	iv
1. INTRODUCTION	1
1.1 Problem Statement	
1.2 Relevance of the Project	
1.3 Objective	
2. SYSTEM REQUIREMENTS	2
2.1 Functional Requirements	
2.2 Hardware and Software Requirements	
3. DESIGN AND IMPLEMENTATION	3-14
3.1 Landing Page & User Authentication	
3.2 Posts & Social Feed	
3.3 Get a Post & Comment Model	
3.4 Edit Post, Delete Post & Add comments	
3.5 Profiles	
3.6 Followers	
3.7 Likes & Dislikes	
3.8 User Search	
3.9 Replying to Comments	
3.10 User Notification	
3.11 Adding Images to Posts	
3.12 Sending & Receiving DMs	
3.13 Sharing Other posts	
4. SCREENSHOTS	15-20
5. CONCLUSION	21
REFERENCES	22

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

Create an online Social Website using REST APIs which include activities like posting, liking, messaging, commenting and sharing.

1.2 RELEVANCE OF THE PROJECT

The website “WE CONNECT” will be used to connect the people in very easy ,simple and efficient way and one can share their feeling ,information ,ideas.....and many more..., the services offered to an individual’s choice(s) and availability for making friends among various areas and destinations. A log concerning the registration and requests for friends and various other features by users are also maintained. The website will also provide benefits to verified user(s).

The website, according to the following proposed solution, will ease the connecting people s thereby converging the world into a small system.

1.3 OBJECTIVE

Social Media is the new trend in today’s time and age. Social media has become an integral part of everyone and is basically considered this generation’s language. Again what is to be noted, is that social media is not only limited to the millennials but people of all ages.

Be it making a Post or uploading one’s moments on Facebook or even Instagram, Or even making a statement or putting down one’s opinion, say on a platform like Twitter, Social media not only has made lives and moments special but also a tool to market, advertise as well as to spread awareness.

Main objective of this project is to connect you with your friends and involve in activities like following friends ,posting ,sharing ,like ,comment ,chat etc..

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 FUNCTIONAL REQUIREMENTS

- User must have a valid User ID and password to login.
- User who don't have their account in this site, can create a new account for signup.
- User should not be allowed to have more than one profile.
- User can edit his/her profile and can post his/her views basing on the activities he/she desires, after the valid user login his/her account.
- User can use the features like auto analyze topics, group discussions, auto recommend threads based on searches etc..

2.2 HARDWARE REQUIREMENTS

The Hardware requirements are very minimal and the program can be run on most of the machines.

- Processor - Intel 486/Pentium processor or better
- Processor Speed - 500 MHz or above
- RAM - 64MB or above
- Storage Space - Approx. 2MB

2.3 SOFTWARE REQUIREMENTS

- Technology Implemented : Django
- Language Used : Python
- User Interface Design : HTML, CSS, JavaScript, BootStrap
- Web Browser : Google Chrome

CHAPTER 3

DESIGN & IMPLEMENTATION

3.1 Landing Page & User Authentication

We are going to start a simple social media website. This website will be pretty similar to Twitter. Users will be able to sign up and post short text as posts and other users will be able to comment on posts. We will have likes and dislikes as well on each post. Users will have the ability to follow other users and see their followers. We will set up authentication with Django AllAuth and we will set up the landing page for guest visitors. We will use Bootstrap 5 which is very new and has some small changes from Bootstrap 4. Most of the classes are the same but there will be some new classes that we will use here.

```
{% extends 'landing/base.html' %}

{% block content %}
<div class="container">
  <div class="row justify-content-center mt-5">
    <div class="col-md-10 col-sm-12 text-center">
      <h1 class="display-2">We Connect</h1>
      <h2>Connect With Your Friends</h2>
      <p class="mt-3 lead">Follow people who interest you, stay up to date on the latest news and join conversations with your friends!</p>
      <div class="d-flex justify-content-center mt-5">
        <a href="{% url 'account_login' %}" class="btn btn-light mr-2">Log In</a>
        <a href="{% url 'account_signup' %}" class="btn btn-dark">Register</a>
      </div>
    </div>
  </div>
</div>
{% endblock content %}
```

Fig:3.1.1

```
{% load custom_tags %}

<div class="container">
<nav class="navbar navbar-expand-lg navbar-light">
  <div class="container-fluid">
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarTogglerDemo03" aria-controls="navbarTogglerDemo03" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <a class="navbar-brand">
      {% if user.is_authenticated %}
        href="{% url 'post-list' %}"
      {% else %}
        href="{% url 'index' %}"
      {% endif %}
    </a>
    <i class="fas fa-comment"></i> We Connect</a>
  </div>
  <div class="collapse navbar-collapse" id="navbarTogglerDemo03">
    <ul class="navbar-nav me-auto mb-2 mb-lg-0">
      <li><a href="{% url 'index' %}">Home</a></li>
      <li><a href="{% url 'post-list' %}">Posts</a></li>
      <li><a href="{% url 'profile' %}">Profile</a></li>
      <li><a href="{% url 'account_logout' %}">Sign Out</a></li>
    </ul>
    <form class="d-flex" method="GET" action="{% url 'profile-search' %}">
      <div class="input-group">
        <span class="input-group-text" id="basic-addon1">@</span>
        <input type="text" class="form-control" placeholder="Username" aria-label="Username" aria-describedby="basic-addon1" name="query" value="{{ request.GET.query }}">
        <button class="remove-default-btn" type="submit"><i class="fas fa-search"></i></button>
      </div>
    </form>
    {% if user.is_authenticated %}
      <div class="nav-item dropdown">
        <a class="nav-link dropdown-toggle text-dark" data-bs-toggle="dropdown" role="button" aria-expanded="false"><i class="fas fa-user"></i></a>
        <ul class="dropdown-menu">
          <li><a class="dropdown-item" href="{% url 'profile' %}">Profile</a></li>
          <li><a class="dropdown-item" href="{% url 'account_logout' %}">Sign Out</a></li>
        </ul>
      </div>
      <div class="nav-item inbox-icon-container">
        <a href="{% url 'inbox' %}" class="inbox-icon"><i class="far fa-paper-plane"></i></a>
      </div>
      <div class="nav-item">
        {% show_notifications %}
      </div>
    </div>
  </nav>
</div>
```

Fig:3.1.2

3.2 Posts & Social Feed

We will add a social feed where posts can be viewed and a form where a new post can be added, we are not going to worry about the small details in what should be shown to each user yet, for now we will just show all posts. We are just creating a simple data model here, where we have a body of text, a date that the post was created on, and which user that wrote the post. Now with that done, we can create a view for the social feed.


```
class PostListView(LoginRequiredMixin, View):
    def get(self, request, *args, **kwargs):
        logged_in_user = request.user
        posts = Post.objects.filter(
            author__profile__followers__in=[logged_in_user.id]
        )
        form = PostForm()
        share_form = ShareForm()

        context = {
            'post_list': posts,
            'shareform': share_form,
            'form': form,
        }

        return render(request, 'social/post_list.html', context)

    def post(self, request, *args, **kwargs):
        logged_in_user = request.user
        posts = Post.objects.filter(
            author__profile__followers__in=[logged_in_user.id]
        )
        form = PostForm(request.POST, request.FILES)
        files = request.FILES.getlist('image')
        share_form = ShareForm()

        if form.is_valid():
            new_post = form.save(commit=False)
            new_post.author = request.user
            new_post.save()

            for f in files:
                img = Image(image=f)
                img.save()
                new_post.image.add(img)

            new_post.save()

        context = {
            'post_list': posts,
            'shareform': share_form,
            'form': form,
        }

        return render(request, 'social/post_list.html', context)
```

Fig:3.2.1 Feed

3.3 Get a Post & Comment Model

We are going to add to the posts feature of our social website by adding the ability to get a single post, and set up the comment model.

```
class Post(models.Model):
    shared_body = models.TextField(blank=True, null=True)
    body = models.TextField()
    image = models.ManyToManyField('Image', blank=True)
    created_on = models.DateTimeField(default=timezone.now)
    shared_on = models.DateTimeField(blank=True, null=True)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    shared_user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True, related_name='+')
    likes = models.ManyToManyField(User, blank=True, related_name='likes')
    dislikes = models.ManyToManyField(User, blank=True, related_name='dislikes')

    class Meta:
        ordering = ['-created_on', '-shared_on']
```

Fig:3.3.1 Post Model

```
class Comment(models.Model):
    comment = models.TextField()
    created_on = models.DateTimeField(default=timezone.now)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    post = models.ForeignKey('Post', on_delete=models.CASCADE)
    likes = models.ManyToManyField(User, blank=True, related_name='comment_likes')
    dislikes = models.ManyToManyField(User, blank=True, related_name='comment_dislikes')
    parent = models.ForeignKey('self', on_delete=models.CASCADE, blank=True, null=True, related_name='+')

    @property
    def children(self):
        return Comment.objects.filter(parent=self).order_by('-created_on').all()

    @property
    def is_parent(self):
        if self.parent is None:
            return True
        return False
```

Fig:3.3.2 Comment Model

3.4 Edit Post, Delete Post & Add Comments

We need to add the ability to edit and delete posts. We also need to make sure to only allow this to the user who originally created the post. After that, we will need to add the ability to add comments and delete comments. We also need to restrict the edit and delete views to be only accessible by the user who created the post/comment.

```
class PostEditView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
    model = Post
    fields = ['body']
    template_name = 'social/post_edit.html'

    def get_success_url(self):
        pk = self.kwargs['pk']
        return reverse_lazy('post-detail', kwargs={'pk': pk})

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

class PostDeleteView(LoginRequiredMixin, UserPassesTestMixin, DeleteView):
    model = Post
    template_name = 'social/post_delete.html'
    success_url = reverse_lazy('post-list')

    def test_func(self):
        post = self.get_object()
        return self.request.user == post.author
```

Fig:3.4.1

3.5 Profiles

We are going to add profiles to our website. We will add a way to view and edit profiles from a link in the navbar. We will also use Django signals to automatically create profiles once a user is created.

```
class ProfileView(View):
    def get(self, request, pk, *args, **kwargs):
        profile = UserProfile.objects.get(pk=pk)
        user = profile.user
        posts = Post.objects.filter(author=user)

        followers = profile.followers.all()

        if len(followers) == 0:
            is_following = False

        for follower in followers:
            if follower == request.user:
                is_following = True
                break
            else:
                is_following = False

        number_of_followers = len(followers)

        context = {
            'user': user,
            'profile': profile,
            'posts': posts,
            'number_of_followers': number_of_followers,
            'is_following': is_following,
        }

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

class ProfileEditView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
    model = UserProfile
    fields = ['name', 'bio', 'birth_date', 'location', 'picture']
    template_name = 'social/profile_edit.html'

    def get_success_url(self):
        pk = self.kwargs['pk']
        return reverse_lazy('profile', kwargs={'pk': pk})

    def test_func(self):
        profile = self.get_object()
        return self.request.user == profile.user
```

Fig:3.5.1

3.6 Followers

We are going to add the ability to follow and unfollow users. We will create two simple views that will handle the logic for this. We will need to update our models.py to have a field to store the followers for each user.

```
class AddFollower(LoginRequiredMixin, View):
    def post(self, request, pk, *args, **kwargs):
        profile = UserProfile.objects.get(pk=pk)
        profile.followers.add(request.user)

        notification = Notification.objects.create(notification_type=3, from_user=request.user, to_user=profile.user)

        return redirect('profile', pk=profile.pk)

class RemoveFollower(LoginRequiredMixin, View):
    def post(self, request, pk, *args, **kwargs):
        profile = UserProfile.objects.get(pk=pk)
        profile.followers.remove(request.user)

        return redirect('profile', pk=profile.pk)
```

Fig:3.6.1

3.7 Likes & Dislikes

We are going to add the ability to like and dislike posts. To do this, we will store a list of users that clicked the like or the dislike button. We will add some checks to make sure that the user can only like or dislike a post once and they can only either like or dislike a post and not both at the same time.

```
class AddLike(LoginRequiredMixin, View):
    def post(self, request, pk, *args, **kwargs):
        post = Post.objects.get(pk=pk)

        is_dislike = False

        for dislike in post.dislikes.all():
            if dislike == request.user:
                is_dislike = True
                break

        if is_dislike:
            post.dislikes.remove(request.user)

        is_like = False

        for like in post.likes.all():
            if like == request.user:
                is_like = True
                break

        if not is_like:
            post.likes.add(request.user)
            notification = Notification.objects.create(notification_type=1, from_user=request.user, to_user=post.author, post=post)

        if is_like:
            post.likes.remove(request.user)

        next = request.POST.get('next', '/')
        return HttpResponseRedirect(next)
```

Fig:3.7.1 Likes

```
class AddDislike(LoginRequiredMixin, View):
    def post(self, request, pk, *args, **kwargs):
        post = Post.objects.get(pk=pk)

        is_like = False

        for like in post.likes.all():
            if like == request.user:
                is_like = True
                break

        if is_like:
            post.likes.remove(request.user)

        is_dislike = False

        for dislike in post.dislikes.all():
            if dislike == request.user:
                is_dislike = True
                break

        if not is_dislike:
            post.dislikes.add(request.user)

        if is_dislike:
            post.dislikes.remove(request.user)

        next = request.POST.get('next', '/')
        return HttpResponseRedirect(next)
```


Fig:3.7.2 Dislikes

3.8 User Search

We are going to get the user search bar that has been in our navbar working. We will use the Q object that comes with Django to get this working. We will need to make some changes to our navbar first.

```
class UserSearch(View):
    def (variable) query: str | None args):|
        query = self.request.GET.get('query')
        profile_list = UserProfile.objects.filter(
            Q(user_username__icontains=query)
        )

        context = {
            'profile_list': profile_list,
        }

        return render(request, 'social/search.html', context)
```

Fig:3.8.1

3.9 Replying to Comments

We will add the ability to like/dislike comments and reply to comments. We won't keep adding replies to replies, etc but we will allow the comments directly on the post to have replies.

```
class CommentReplyView(LoginRequiredMixin, View):
    def post(self, request, post_pk, pk, *args, **kwargs):
        post = Post.objects.get(pk=post_pk)
        parent_comment = Comment.objects.get(pk=pk)
        form = CommentForm(request.POST)

        if form.is_valid():
            new_comment = form.save(commit=False)
            new_comment.author = request.user
            new_comment.post = post
            new_comment.parent = parent_comment
            new_comment.save()

            notification = Notification.objects.create(notification_type=2, from_user=request.user, to_user=parent_comment.author, comment=new_comment)

            return redirect('post-detail', pk=post_pk)
```

Fig:3.9.1

3.10 User Notification

We are going to add user notifications to our social media website. We will set up notifications that will be in the navigation bar that will list all unseen notifications. For our website, we will count a notification as unseen if they haven't clicked on it. This will definitely be more involved and not as simple as the previous changes we have made. To make this all work we will mix in a little Javascript and custom Django template tags.

```
class PostNotification(View):
    def get(self, request, notification_pk, post_pk, *args, **kwargs):
        notification = Notification.objects.get(pk=notification_pk)
        post = Post.objects.get(pk=post_pk)

        notification.user_has_seen = True
        notification.save()

        return redirect('post-detail', pk=post_pk)

class FollowNotification(View):
    def get(self, request, notification_pk, profile_pk, *args, **kwargs):
        notification = Notification.objects.get(pk=notification_pk)
        profile = UserProfile.objects.get(pk=profile_pk)

        notification.user_has_seen = True
        notification.save()

        return redirect('profile', pk=profile_pk)

class ThreadNotification(View):
    def get(self, request, notification_pk, object_pk, *args, **kwargs):
        notification = Notification.objects.get(pk=notification_pk)
        thread = ThreadModel.objects.get(pk=object_pk)

        notification.user_has_seen = True
        notification.save()

        return redirect('thread', pk=object_pk)

class RemoveNotification(View):
    def delete(self, request, notification_pk, *args, **kwargs):
        notification = Notification.objects.get(pk=notification_pk)

        notification.user_has_seen = True
        notification.save()

        return HttpResponseRedirect('Success', content_type='text/plain')
```

Fig:3.10.1

3.11 Adding Images to Posts

We are going to add more to our social network application. This time we will add the ability to upload a image in a post. We will add a button to select a file and then we will show it above the text of the post. We will need to make some updates to our PostModel to handle this.

```
class PostListView(LoginRequiredMixin, View):
    def get(self, request, *args, **kwargs):
        logged_in_user = request.user
        posts = Post.objects.filter(
            author__profile__followers__in=[logged_in_user.id]
        )
        form = PostForm()
        share_form = ShareForm()

        context = {
            'post_list': posts,
            'shareform': share_form,
            'form': form,
        }

        return render(request, 'social/post_list.html', context)

    def post(self, request, *args, **kwargs):
        logged_in_user = request.user
        posts = Post.objects.filter(
            author__profile__followers__in=[logged_in_user.id]
        )
        form = PostForm(request.POST, request.FILES)
        files = request.FILES.getlist('image')
        share_form = ShareForm()

        if form.is_valid():
            new_post = form.save(commit=False)
            new_post.author = request.user
            new_post.save()

            for f in files:
                img = Image(image=f)
                img.save()
                new_post.image.add(img)

            new_post.save()

        context = {
            'post_list': posts,
            'shareform': share_form,
            'form': form,
        }

        return render(request, 'social/post_list.html', context)
```

Fig:3.11.1

3.12 Sending & Receiving DMs

We are going to start to add direct messages between users. In this part we will get it working with its basic functionality and then we will come back and add some extra features to it next. This will not be a real time chat, to do that would be more complicated and would require something like Django channels.


```
class CreateMessage(View):
    def post(self, request, pk, *args, **kwargs):
        form = MessageForm(request.POST, request.FILES)
        thread = ThreadModel.objects.get(pk=pk)
        if thread.receiver == request.user:
            receiver = thread.user
        else:
            receiver = thread.receiver

        if form.is_valid():
            message = form.save(commit=False)
            message.thread = thread
            message.sender_user = request.user
            message.receiver_user = receiver
            message.save()

            notification = Notification.objects.create(
                notification_type=4,
                from_user=request.user,
                to_user=receiver,
                thread=thread
            )
            return redirect('thread', pk=pk)
```

Fig:3.12.1 Sending Message

```
class ListThreads(View):
    def get(self, request, *args, **kwargs):
        threads = ThreadModel.objects.filter(Q(user=request.user) | Q(receiver=request.user))

        context = {
            'threads': threads
        }

        return render(request, 'social/inbox.html', context)
```

Fig:3.12.2 Chat List

```
class ThreadView(View):
    def get(self, request, pk, *args, **kwargs):
        form = MessageForm()
        thread = ThreadModel.objects.get(pk=pk)
        message_list = MessageModel.objects.filter(thread__pk__contains=pk)
        context = {
            'thread': thread,
            'form': form,
            'message_list': message_list
        }

        return render(request, 'social/thread.html', context)
```

Fig:3.12.3 Conversation

3.13 Sharing Other posts

We will be adding the ability to share another users post. To do this we will add more fields to our Post model and we will update our forms, views and templates to handle this new feature.

```
class SharedPostView(View):
    def post(self, request, pk, *args, **kwargs):
        original_post = Post.objects.get(pk=pk)
        form = ShareForm(request.POST)

        if form.is_valid():
            new_post = Post(
                shared_body=self.request.POST.get('body'),
                body=original_post.body,
                author=original_post.author,
                created_on=original_post.created_on,
                shared_user=request.user,
                shared_on=timezone.now(),
            )
            new_post.save()

            for img in original_post.image.all():
                new_post.image.add(img)

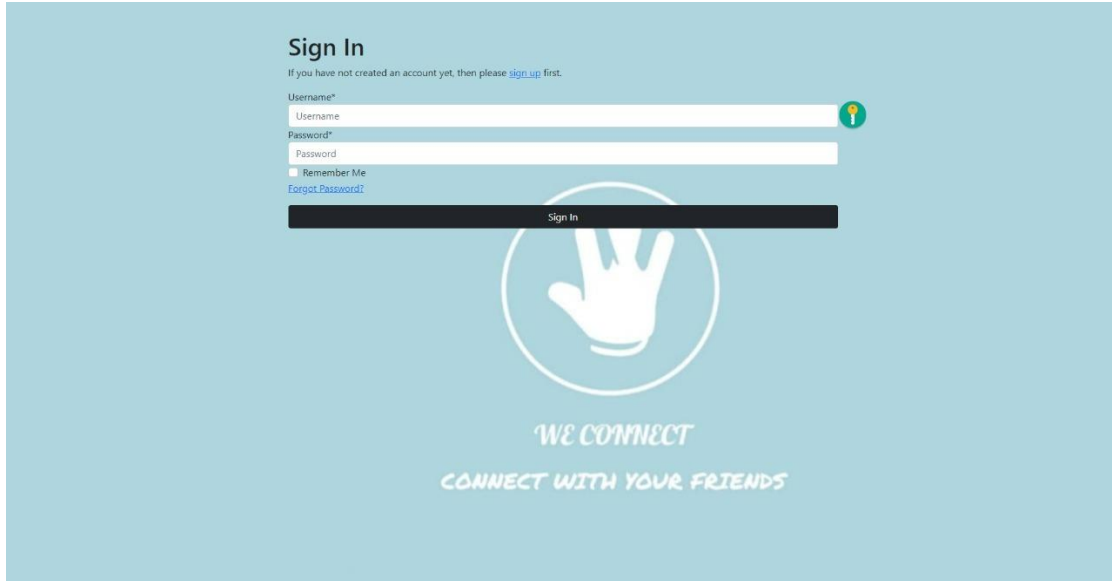
            new_post.save()

        return redirect('post-list')
```

Fig:3.13.1

CHAPTER 4

SCREENSHOTS



Sign In

If you have not created an account yet, then please [sign up](#) first.

Username*
Username

Password*
Password

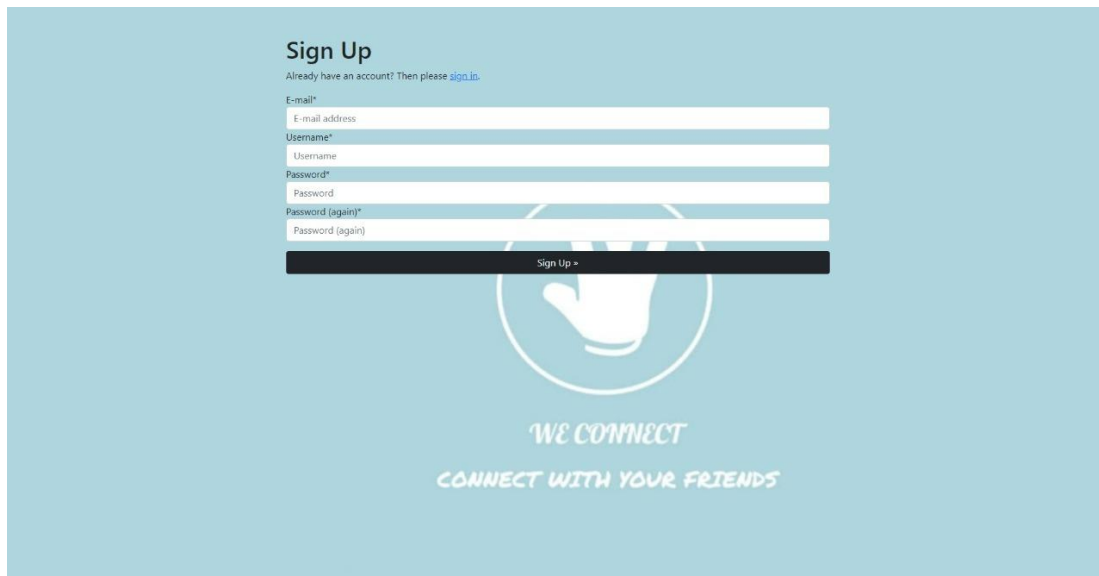
☐ Remember Me

[Forgot Password?](#)

Sign In

WE CONNECT
CONNECT WITH YOUR FRIENDS

Fig:4.1 Sign in



Sign Up

Already have an account? Then please [sign in](#).

E-mail*
E-mail address

Username*
Username

Password*
Password

Password (again)*
Password (again)

Sign Up »

WE CONNECT
CONNECT WITH YOUR FRIENDS

Fig:4.2 Sign up

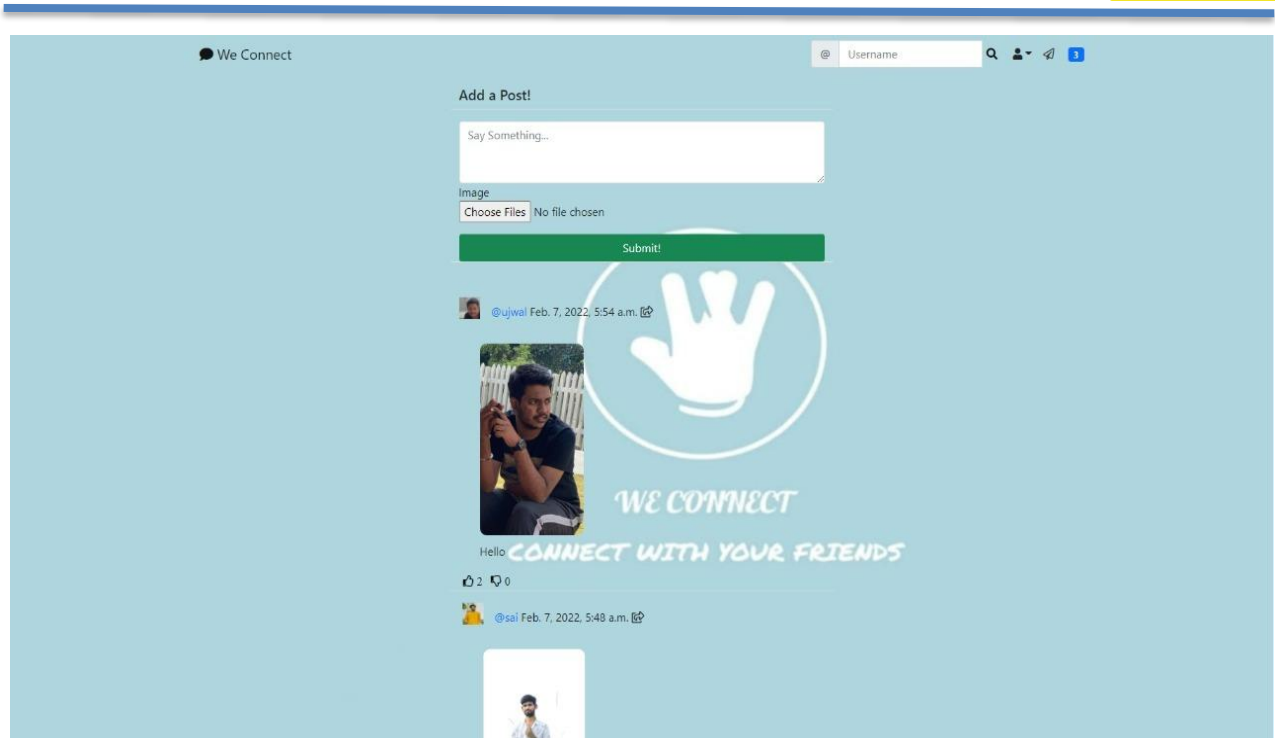


Fig:4.3 Feed

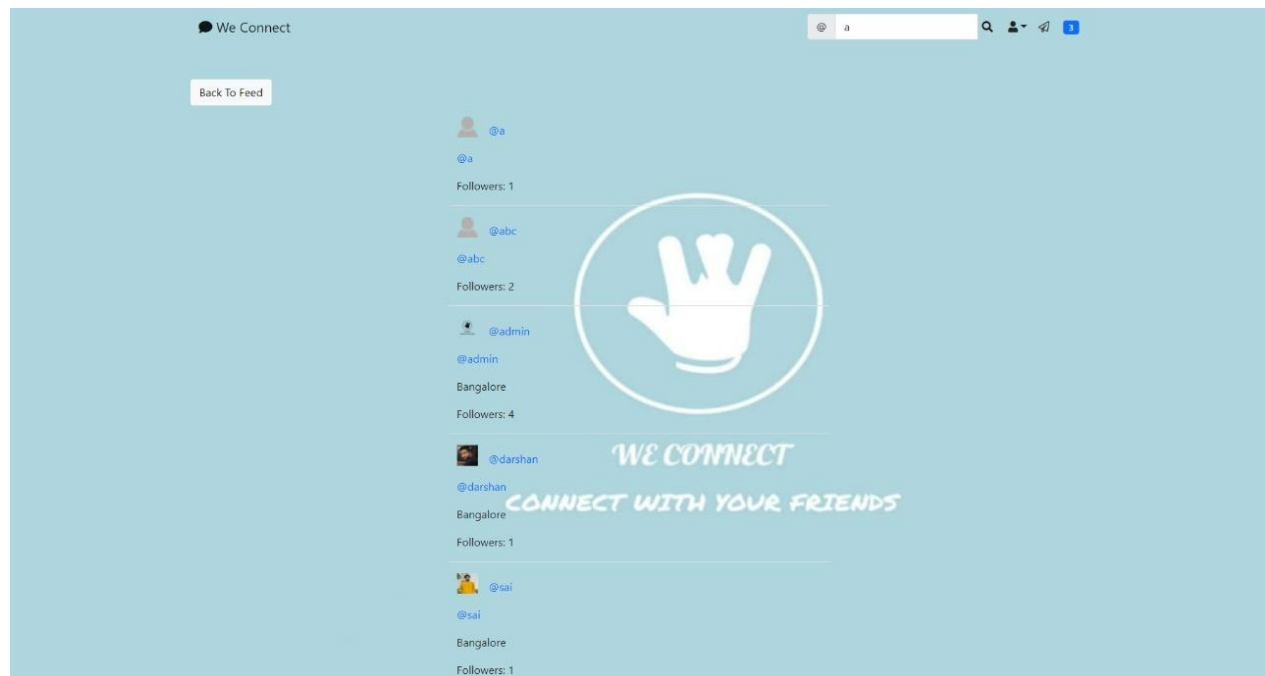


Fig:4.4 user search

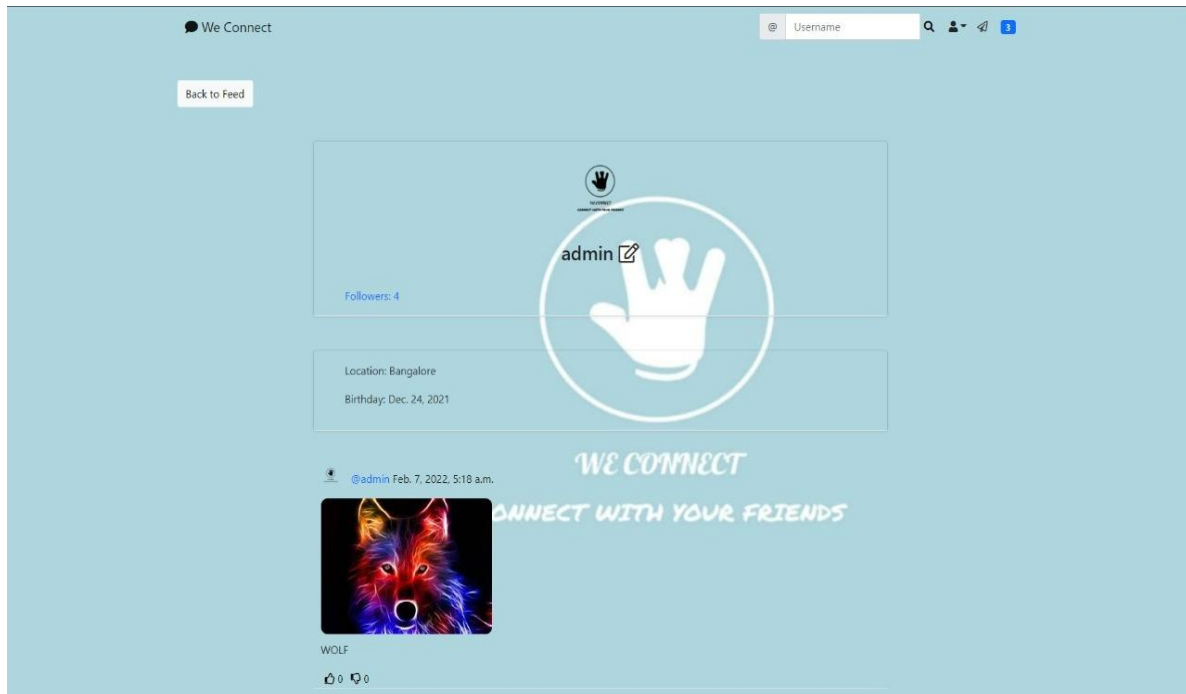


Fig:4.5 profile

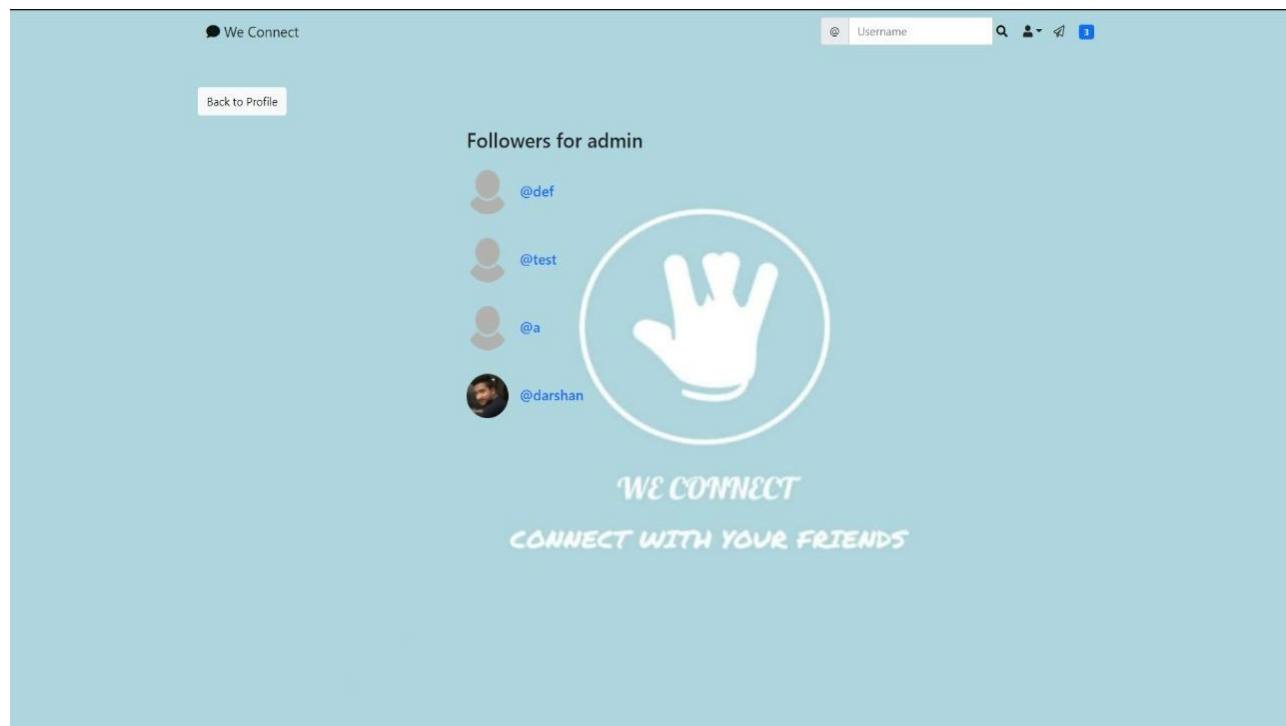


Fig:4.6 Follower list

The screenshot shows the 'Update Your Profile' form in the We Connect application. The form is set against a light blue background. At the top left, there is a 'Back To Profile' button. The form fields include: 'Name' with the value 'admin', 'Bio' with a large empty text area, 'Birth date' with the value '2021-12-24', 'Location' with the value 'Bangalore', and 'Picture' with a preview of 'uploads/profile_pictures/background.png' and a 'Clear' button. Below the picture field is a 'Change:' label with a 'Choose File' button and the text 'No file chosen'. A green 'Submit!' button is at the bottom of the form. The top navigation bar includes the 'We Connect' logo, a search bar with the placeholder 'Username', and icons for user profile, notifications, and a message count of 1. A faint watermark 'CONNECT WITH YOUR FRIENDS' is visible in the background.

Fig:4.7 Editing Profile

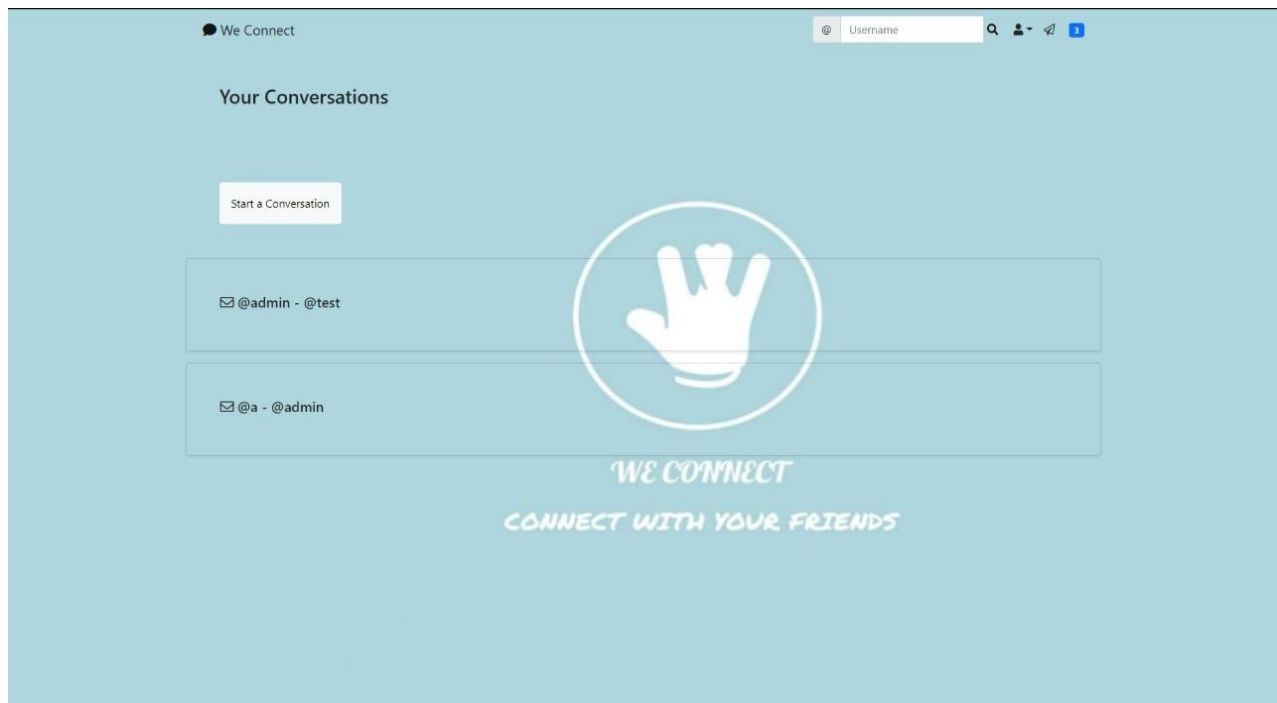


Fig:4.8 Chat List

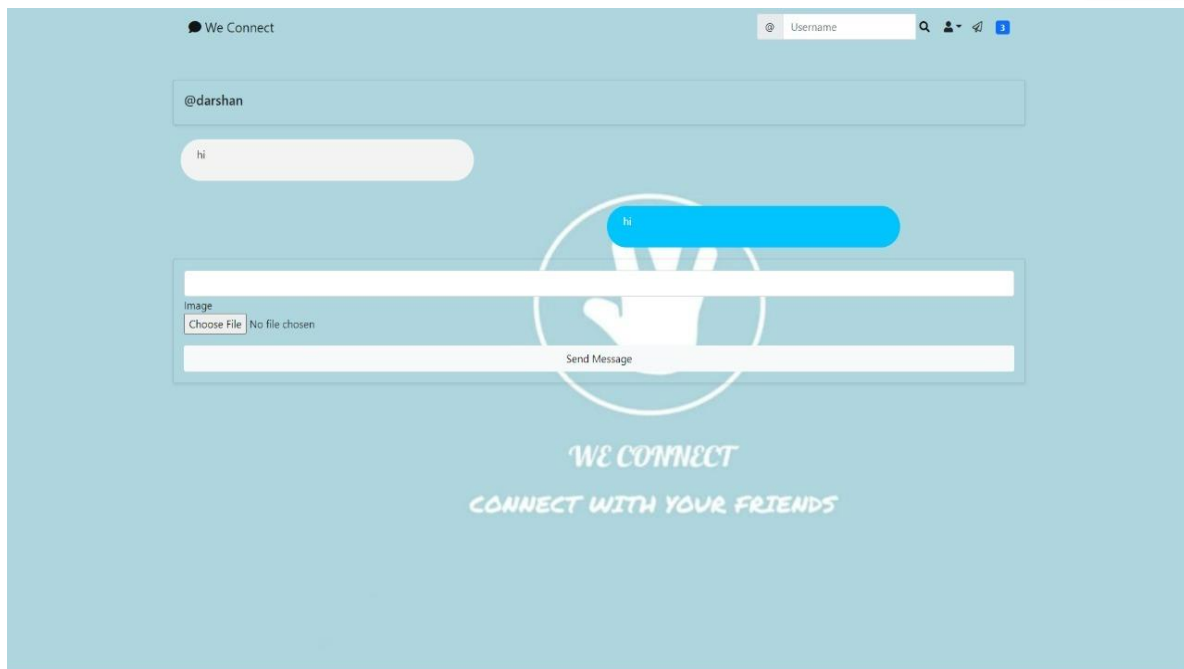


Fig:4.9 Conversation

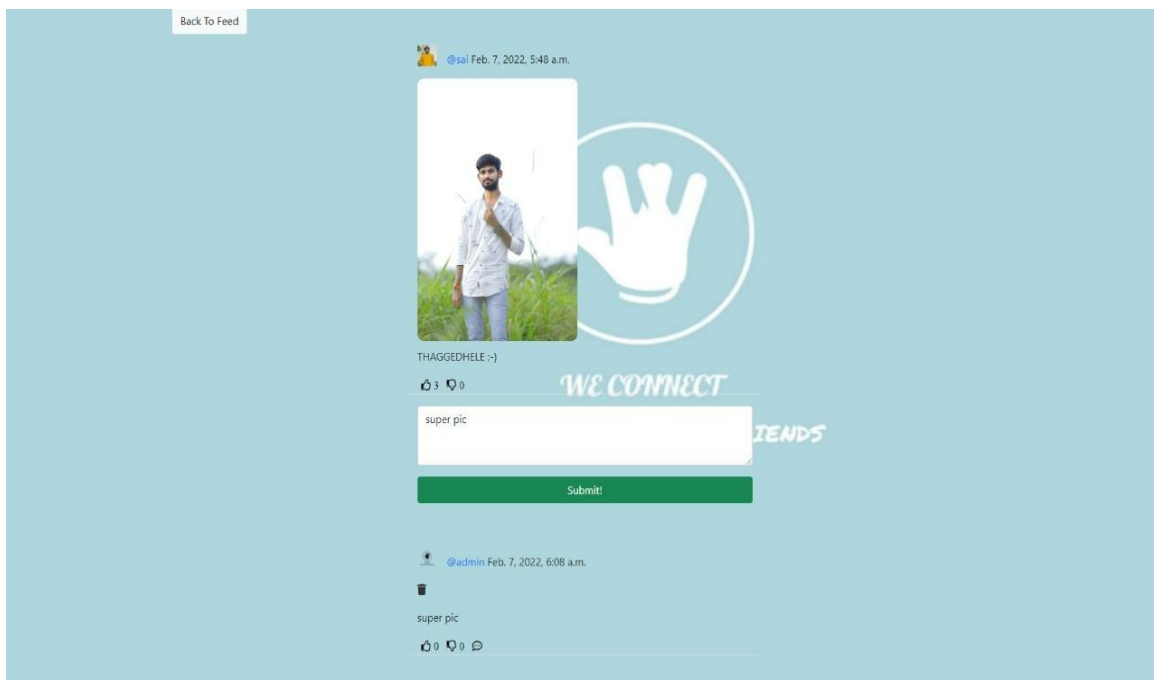


Fig:4.10 Comments



Fig:4.11 Sharing other posts



Fig:4.12 Editing the post

CHAPTER 5

CONCLUSION

Social media is a really convenient and important communicate network for all the people nowadays. We can use it to know friends and keep contact with friends that came from different countries. We can also share our ideas so quickly so that all the things could develop so fast because people could tell us their ideas and we could improve it immediately. There are more advantages for using social media, however, there is always advantages and disadvantages for a thing. As social media is too convenient for people, almost most of them don't even have to 'speak out' to communicate with people. No longer, people will lost their communication skills.

This project is designed to meet the requirements of Social Networking Site. It has been developed using Python Rest APIs keeping in mind the specification of the system.

For designing the system we have used HTML, Python, Django.

Overall the project teaches us the essential skills like:

- Website designing using HTML, CSS, JavaScript, Bootstrap.
- Coding in Python, Introduced with some modules like Django

REFERENCES

- ✓ <https://www.youtube.com/playlist?list=PLPSM8rId1a3TkwEmHyDALNuHhqiUiU5A>
- ✓ <https://github.com/legionscript/socialnetwork/tree/master>
- ✓ <http://legionscript.com/articles>