Dharmsinh Desai University, Nadiad

Faculty of Technology

Department of Computer Engineering

B. Tech. CE Semester – IV

Subject: SEPP & SP

Project Title:

<u>ConnectUs</u>

By

1. Manan Goria, Roll No :- CE042, ID – 19CEUOS086
2. Dipak Gorfad, Roll No :- CE041, ID – 19CEUBG115

Guided by:
Prof. Brijesh Bhatt,
Prof. Jigar Pandya,
Prof. Pinkal Chauhan

1. Abstract

ConnectUs is a social Media Web developed using Django python full-stack web framework

ConnectUs was created using Django and other tools such that it supports all features which are provided by Web chat applications. ConnectUs is easy to use and highly flexible with the user. New user needs to create account and login to respective user id. After successfully logged in user will be taken to home page where the user can add room name to enter in that chat room. User can edit their personal details, password and they get feature to reset their password using email they have register. User get feature of searching other user's profile.

GitHub repository Link: https://github.com/manan2171/ConnectUs.git

2. Tools And Technologies Used

Technologies:

- Django
- Python
- SQLite
- Bootstrap
- JavaScript
- CSS
- HTML

TOOL:

- Git
- Visual Studio Code

3.Software Requirement Specifications

1. Product Scope :

   This system is designed to enable the user to connect with other people via chat.

2. System Functional Requirements:

R.1: Signup

Description: New users have to create an account in order to access ConnectUs web.

INPUT: username, Email_id, Password1, Password2

PROCESSING: Check if username, email is already exist in Database. System also check if Password1 and Password2 are same as well as also check if Password is not similar to username or email.

OUTPUT: User will be created in database.

NEXT: Home Page

R.2: Login

Description: Users who already have account need to login by entering username and password correctly.

R.2.1: Login

State: User is already signed up.

INPUT: username and password.

PROCESSING: Check if username is exist in database.

Then check if password associated with that username is same as entered by user.

OUTPUT: Takes user to Home page.

NEXT: Home Page

R.2.2: ResetPassword

STATE: User must have account created

INPUT: Email of user

OUTPUT: R.5

R.3: Profile:

Description: Once User have created account and login in User can see profile. Profile contains information About user’s email id , username , last time user have Logged in and user’s profile picture.

R.3.1: Profile View:

STATE: User must be logged in.

INPUT: Clicking on Profile option

OUTPUT: Takes User to his Profile Page.

R.3.2: Search User

STATE: User must be logged in.

INPUT: username which user wants to search for

OUTPUT: All users with username entered by user

R.3.3: Edit Profile

STATE: User must be logged in.

INPUT: username or email which user wants to change

OUTPUT: data of user will be changed.

R.3.4: Edit Password:

STATE: User must be logged in.

INPUT: Enter Old Password, New Password and Re-enter New Password.

OUTPUT: Password will be changed.

R.4: Chat Room

Description: User will have to enter room name at input field given in home page. User will be redirected to that chat room page where user can chat with his/her friend.

R.4.1: Creating/Entering Chat room:

STATE: User must be logged in.

INPUT: enter room name

OUTPUT: takes user to that chat room

R.4.2: Type and send massage

STATE: User must be logged in and user must have entered in chat room.

INPUT: Type message in input field.

OUTPUT: Message will pop in chat box.

R.5: Password Reset:

Description: User can reset password from login page If password have been forgotten. User will need to enter email. User will get email from our system to reset password.

R.5.1: Entering email:

STATE: User must have account registered.

INPUT: Entering Email in form.

OUTPUT: Receives email from system.

R.5.2: Password Reset form:

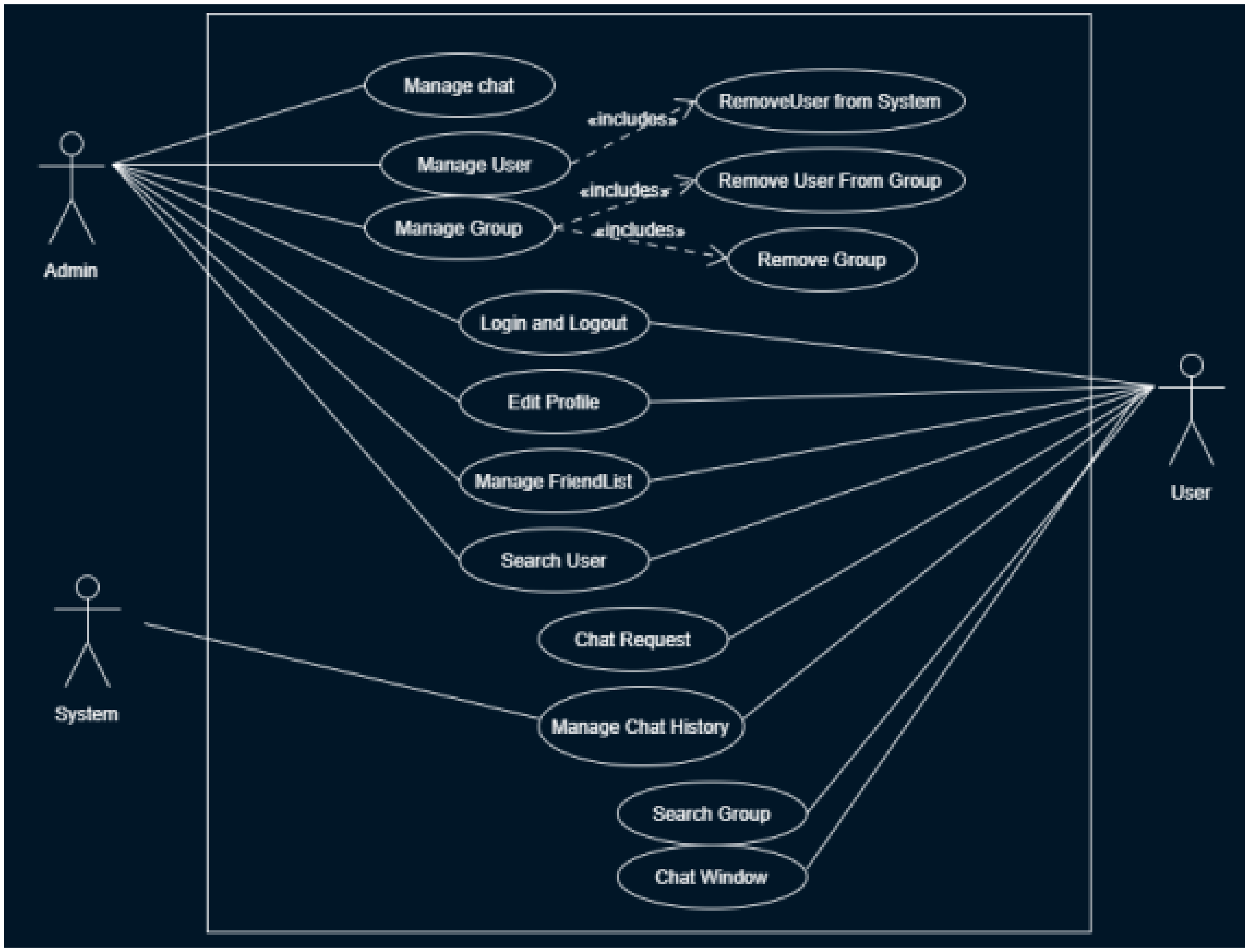
STATE: User must have account registered.

INPUT: Entering new password, Re-entering New Password.

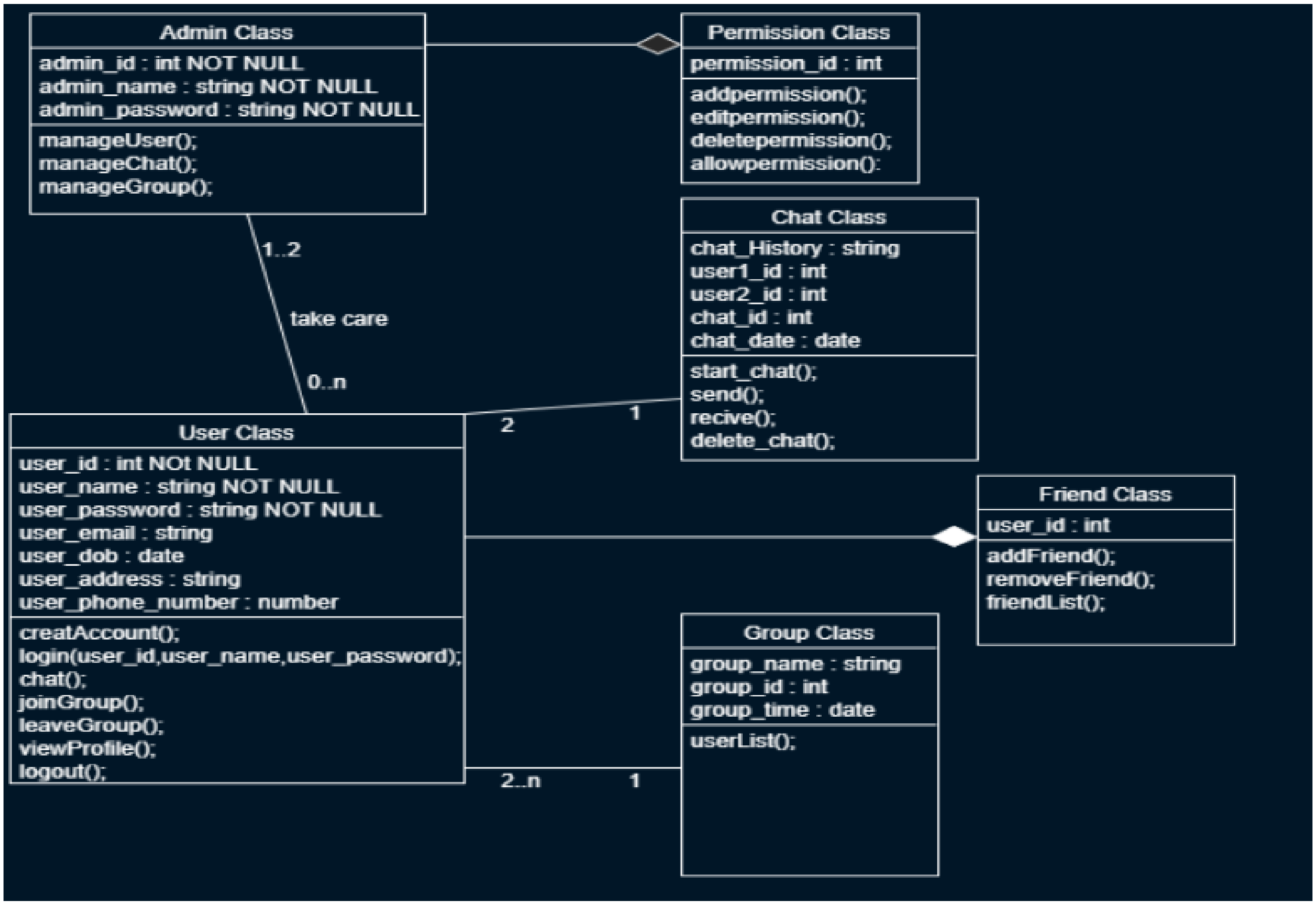
OUTPUT: Password will be Changed.

4. Design

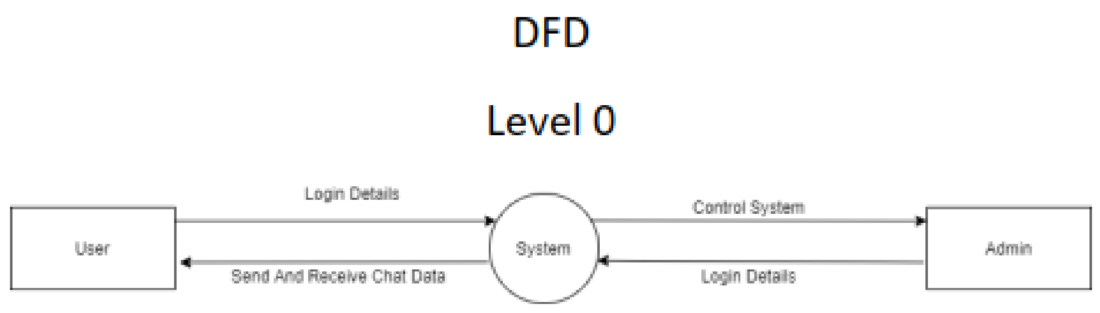
4.1 Use case Diagram



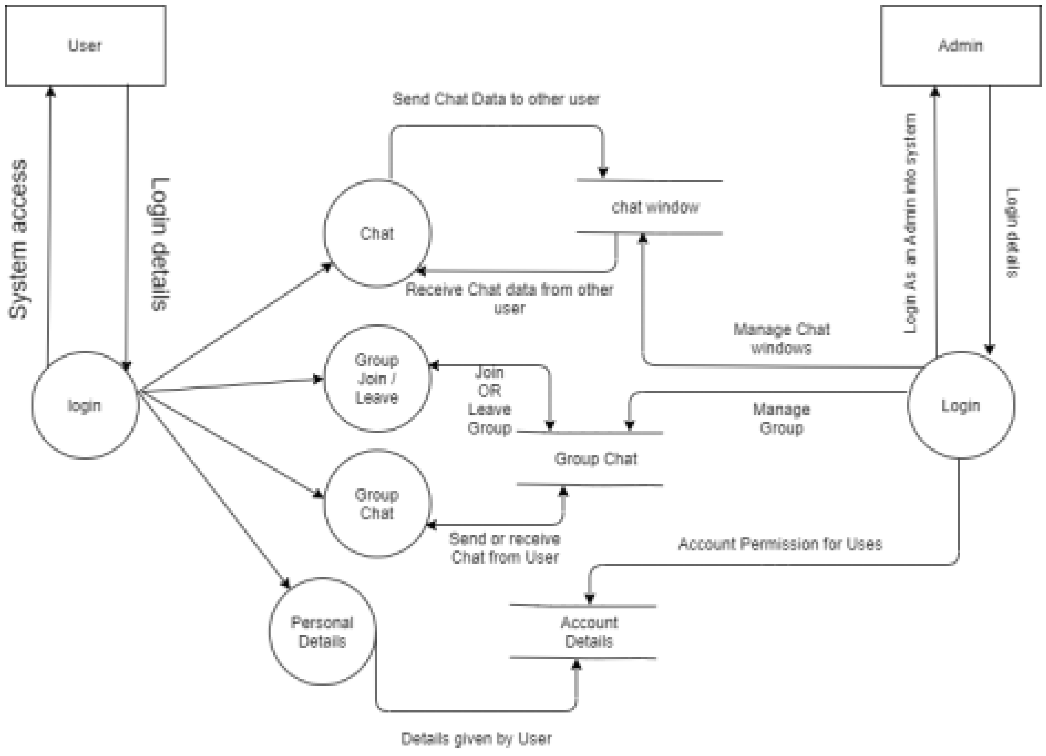
4.2CLASS DIAGRAM:



4.3DFD Diagram:

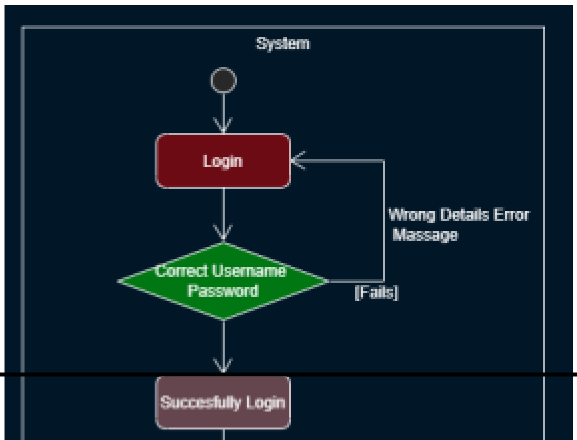


Level 1

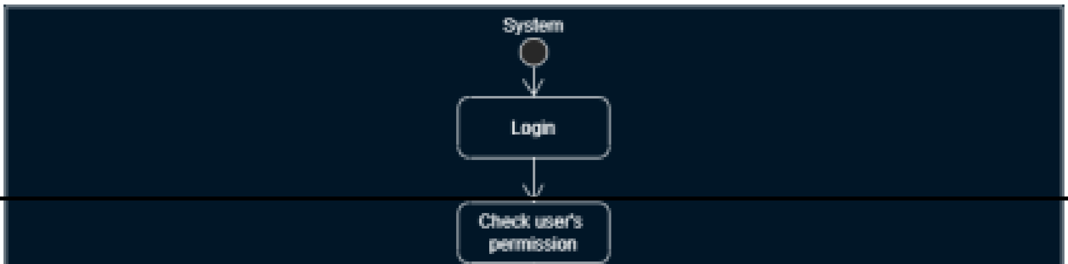


4.4Activity Diagram:

For Login Class:

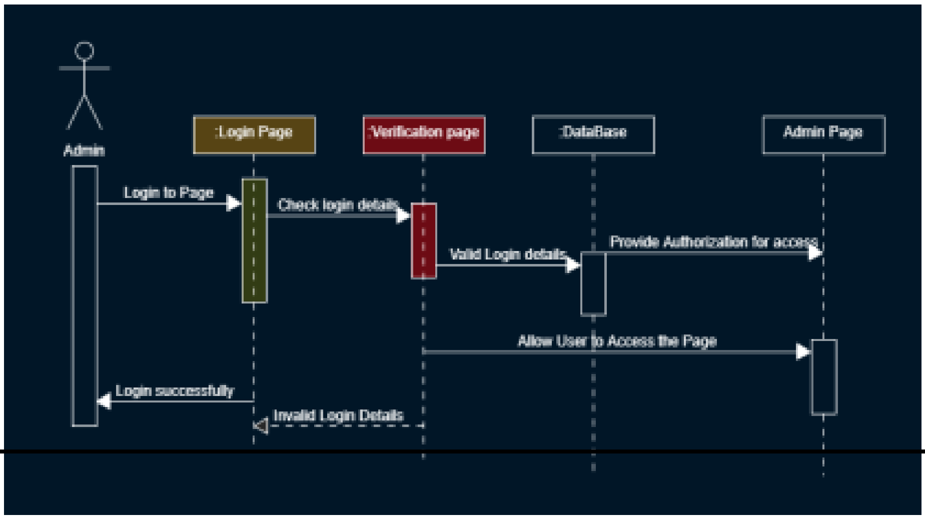


For Management Class

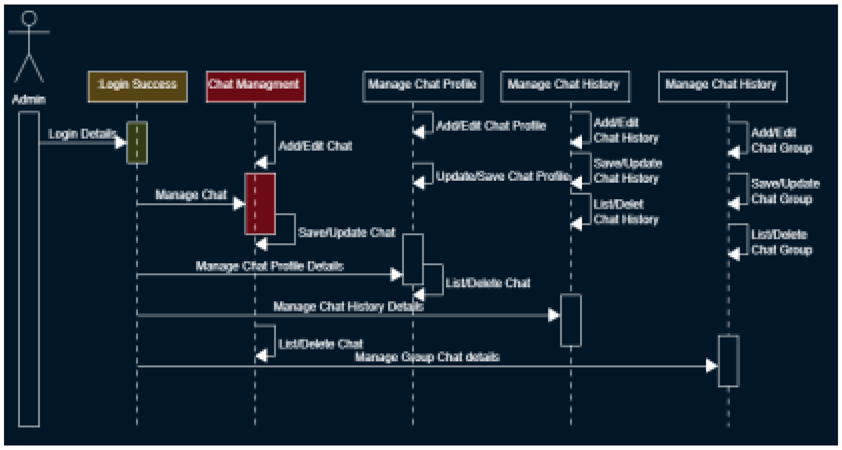


4.5 Sequence Diagram:

For Login Class:

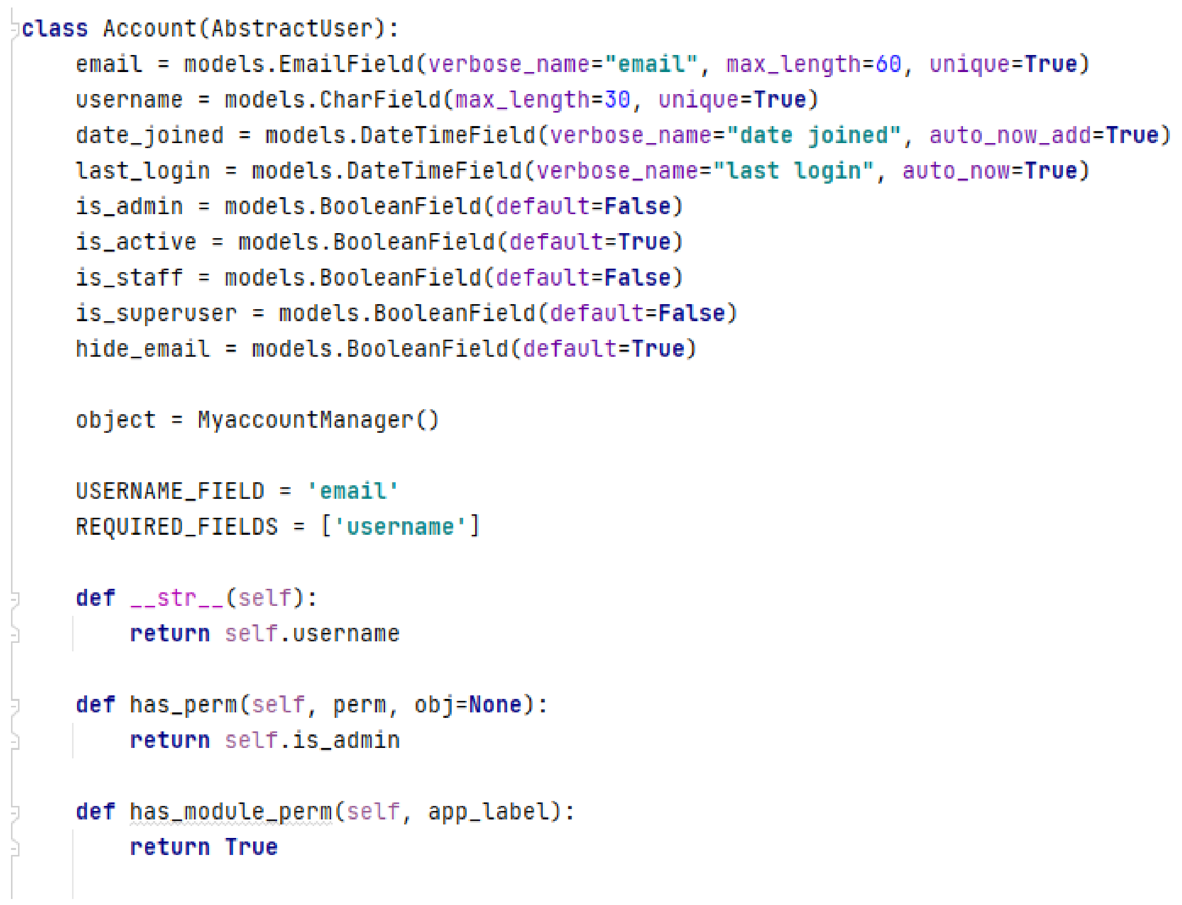


For Manage Class



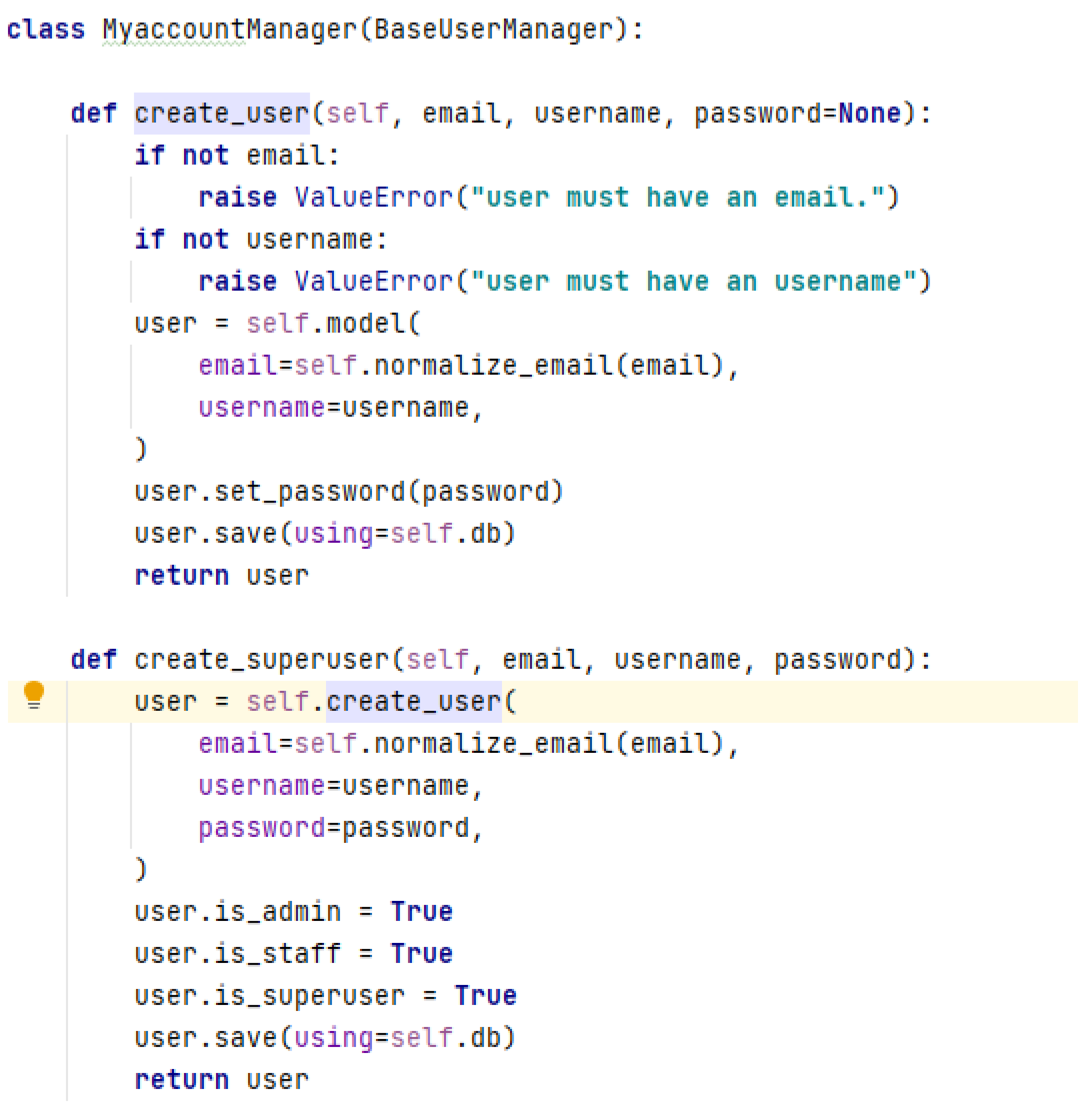
5. ConnectUs Django Models

User Model



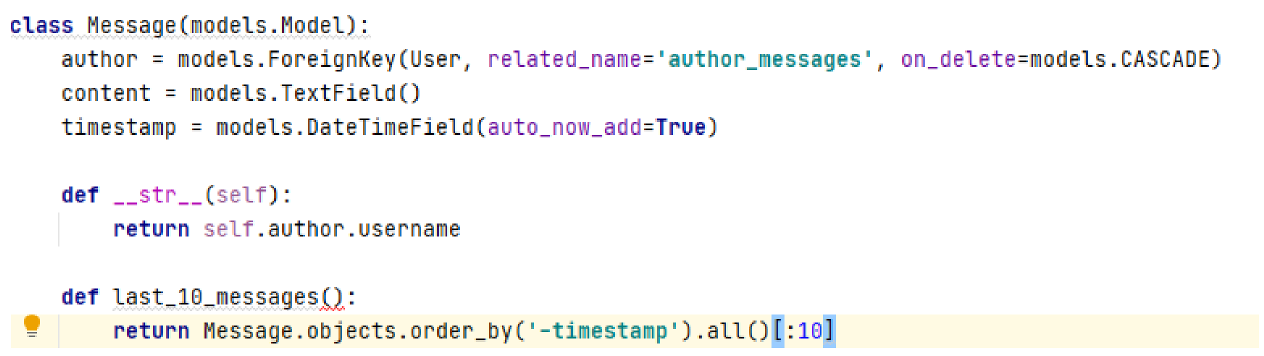
```python
class Account(AbstractUser):
    email = models.EmailField(verbose_name="email", max_length=60, unique=True)
    username = models.CharField(max_length=30, unique=True)
    date_joined = models.DateTimeField(verbose_name="date joined", auto_now_add=True)
    last_login = models.DateTimeField(verbose_name="last login", auto_now=True)
    is_admin = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)
    hide_email = models.BooleanField(default=True)

    object = MyaccountManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username']

    def __str__(self):
        return self.username

    def has_perm(self, perm, obj=None):
        return self.is_admin

    def has_module_perm(self, app_label):
        return True
```

User creation Model



```python
class MyaccountManager(BaseUserManager):

    def create_user(self, email, username, password=None):
        if not email:
            raise ValueError("user must have an email.")
        if not username:
            raise ValueError("user must have an username")
        user = self.model(
            email=self.normalize_email(email),
            username=username,
        )
        user.set_password(password)
        user.save(using=self.db)
        return user

    def create_superuser(self, email, username, password):
        user = self.create_user(
            email=self.normalize_email(email),
            username=username,
            password=password,
        )
        user.is_admin = True
        user.is_staff = True
        user.is_superuser = True
        user.save(using=self.db)
        return user
```

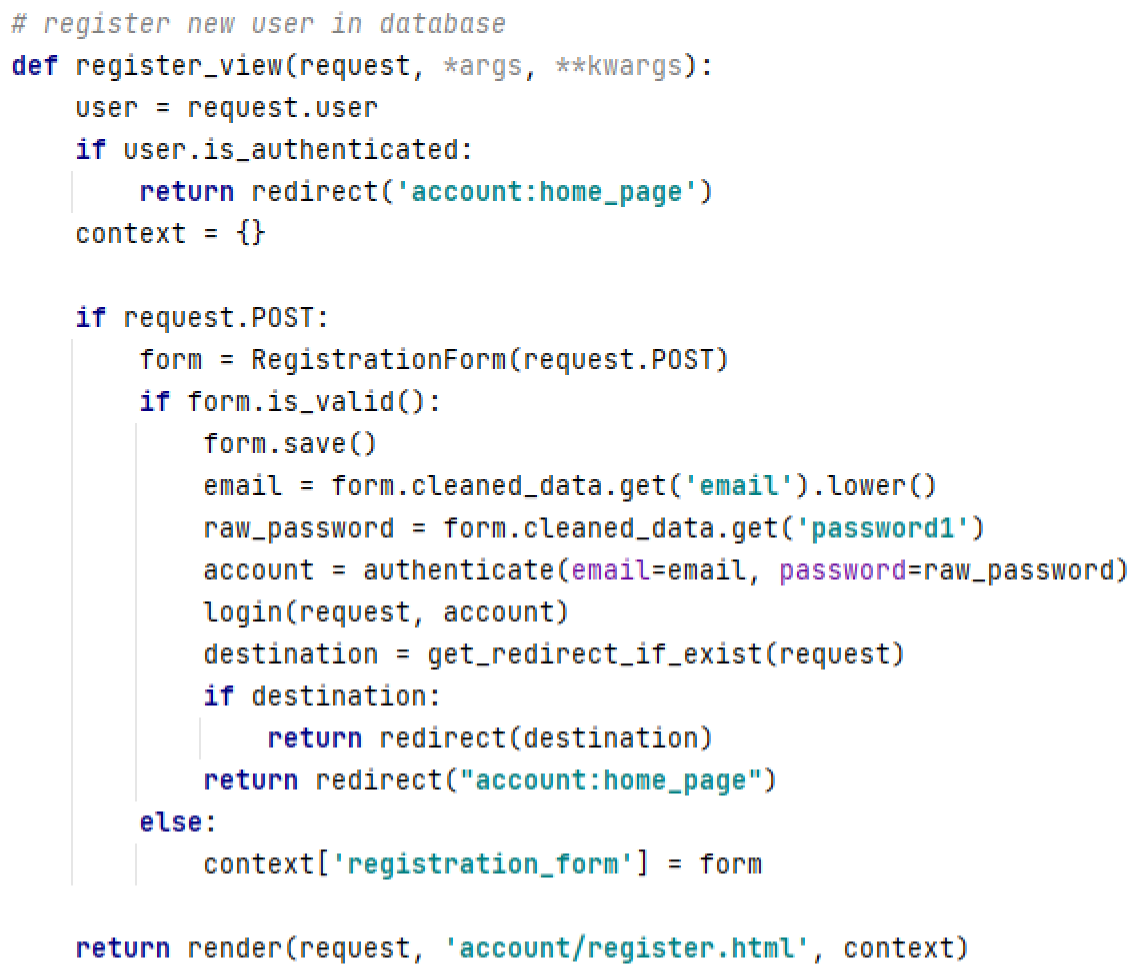
Message Model:



```python
class Message(models.Model):
    author = models.ForeignKey(User, related_name='author_messages', on_delete=models.CASCADE)
    content = models.TextField()
    timestamp = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.author.username

    def last_10_messages():
        return Message.objects.order_by('-timestamp').all()[:10]
```

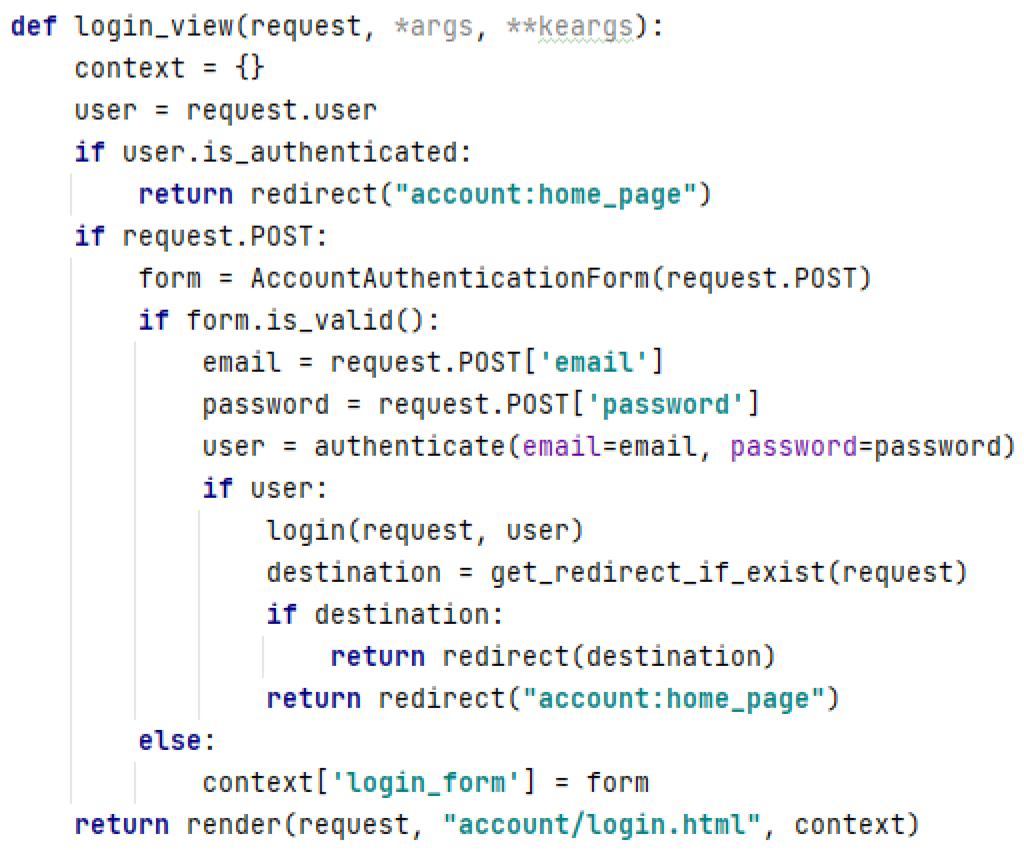
6. ConnectUs Django Views

SignUp View:



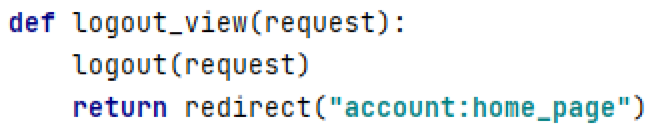
```python
# register new user in database
def register_view(request, *args, **kwargs):
    user = request.user
    if user.is_authenticated:
        return redirect('account:home_page')
    context = {}

    if request.POST:
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            email = form.cleaned_data.get('email').lower()
            raw_password = form.cleaned_data.get('password1')
            account = authenticate(email=email, password=raw_password)
            login(request, account)
            destination = get_redirect_if_exist(request)
            if destination:
                return redirect(destination)
            return redirect("account:home_page")
        else:
            context['registration_form'] = form

    return render(request, 'account/register.html', context)
```

Login View:



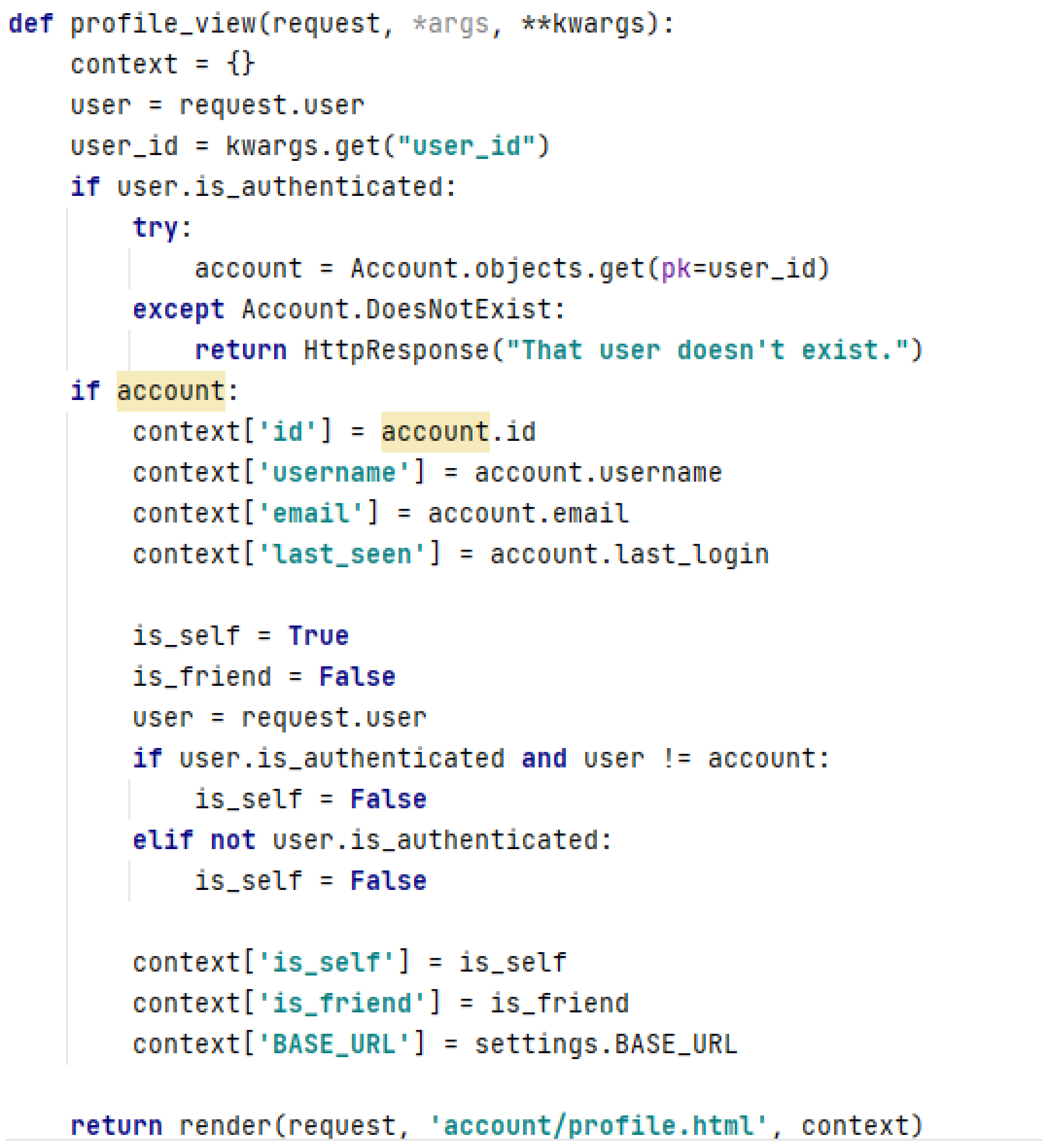
```python
def login_view(request, *args, **keargs):
    context = {}
    user = request.user
    if user.is_authenticated:
        return redirect("account:home_page")
    if request.POST:
        form = AccountAuthenticationForm(request.POST)
        if form.is_valid():
            email = request.POST['email']
            password = request.POST['password']
            user = authenticate(email=email, password=password)
            if user:
                login(request, user)
                destination = get_redirect_if_exist(request)
                if destination:
                    return redirect(destination)
                return redirect("account:home_page")
        else:
            context['login_form'] = form
    return render(request, "account/login.html", context)
```

Logout View:



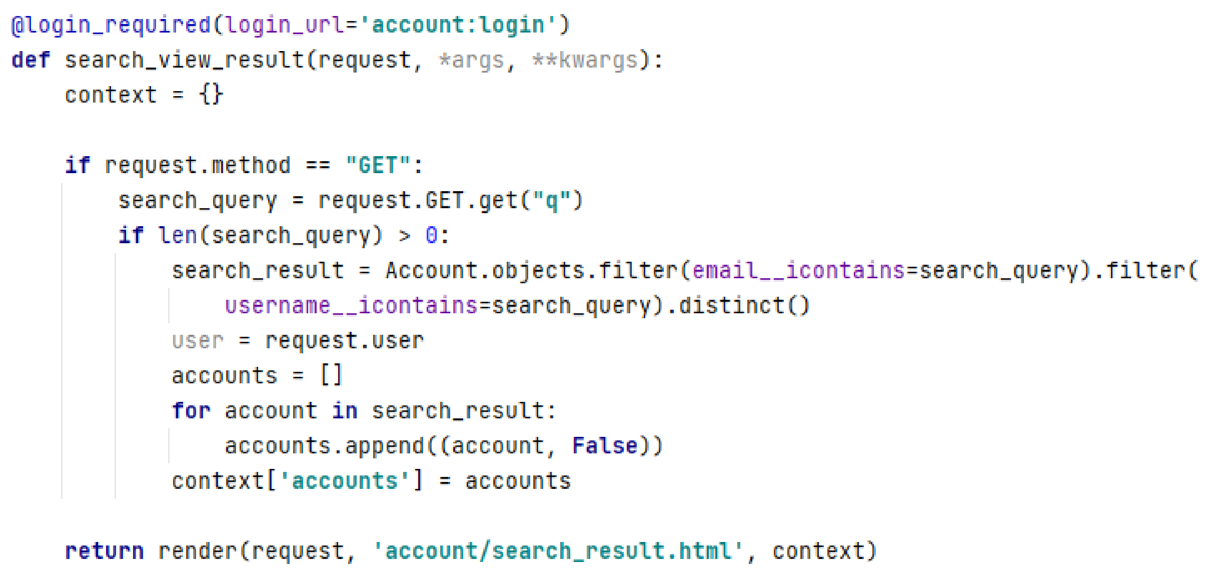
```python
def logout_view(request):
    logout(request)
    return redirect("account:home_page")
```

Profile View:



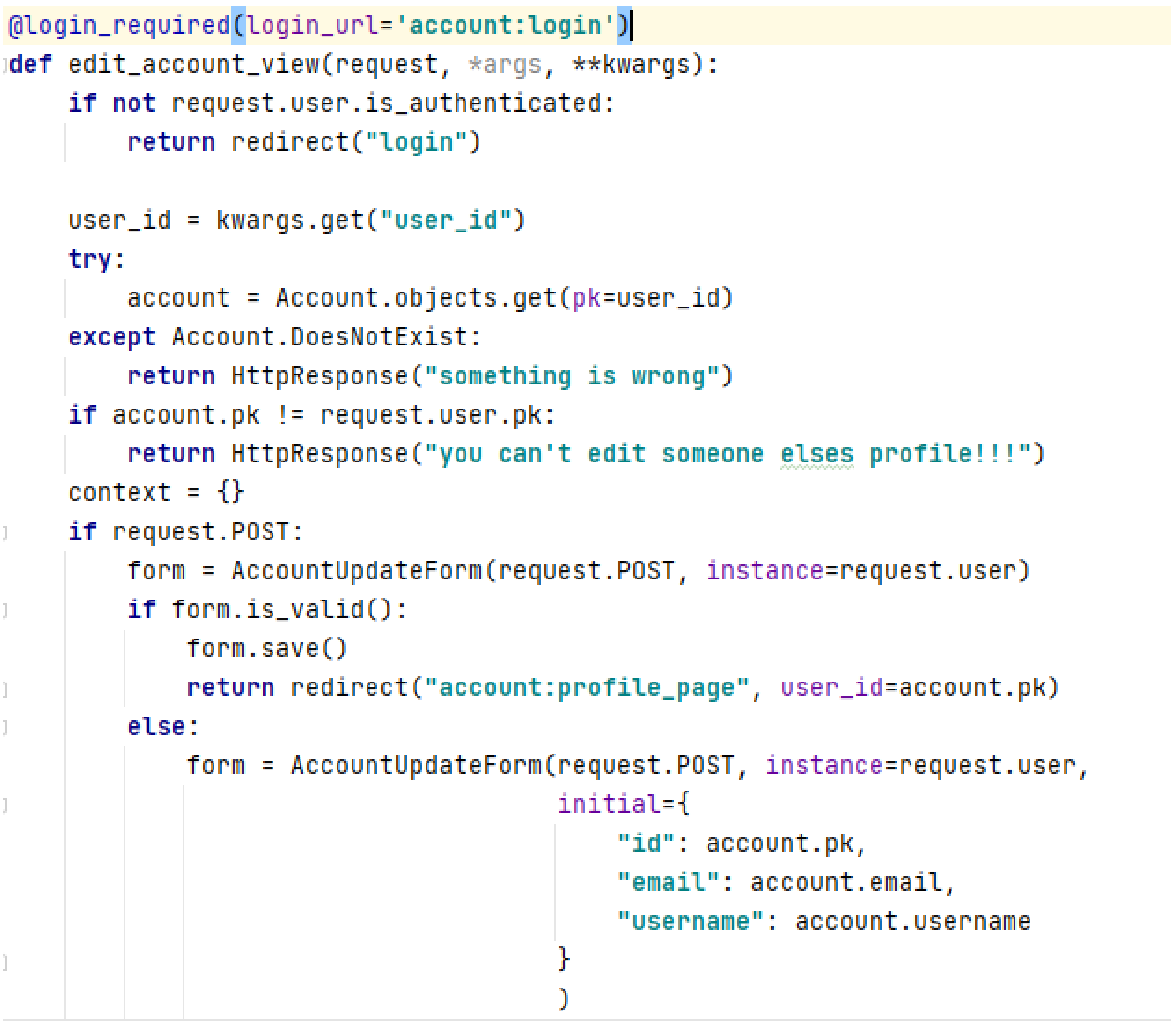
```python
def profile_view(request, *args, **kwargs):
    context = {}
    user = request.user
    user_id = kwargs.get("user_id")
    if user.is_authenticated:
        try:
            account = Account.objects.get(pk=user_id)
        except Account.DoesNotExist:
            return HttpResponse("That user doesn't exist.")
    if account:
        context['id'] = account.id
        context['username'] = account.username
        context['email'] = account.email
        context['last_seen'] = account.last_login

        is_self = True
        is_friend = False
        user = request.user
        if user.is_authenticated and user != account:
            is_self = False
        elif not user.is_authenticated:
            is_self = False

        context['is_self'] = is_self
        context['is_friend'] = is_friend
        context['BASE_URL'] = settings.BASE_URL

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

Search View:

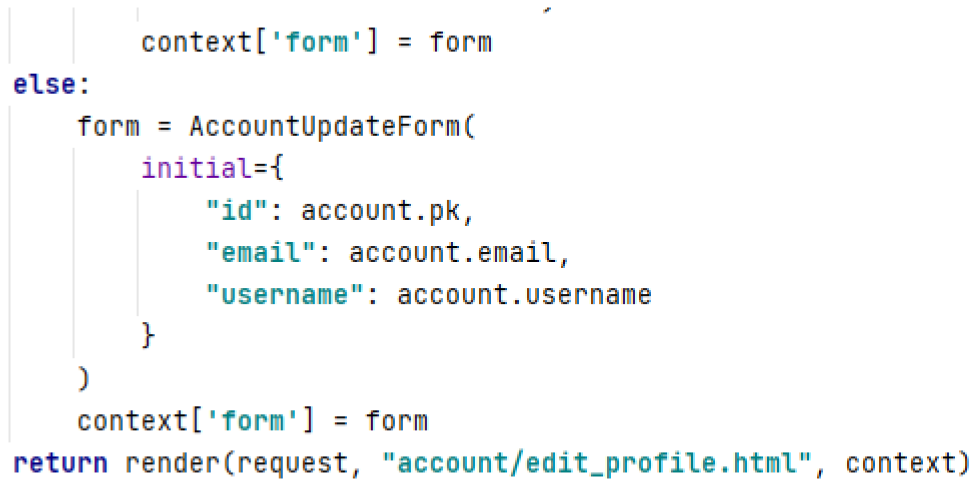


```python
@login_required(login_url='account:login')
def search_view_result(request, *args, **kwargs):
    context = {}

    if request.method == "GET":
        search_query = request.GET.get("q")
        if len(search_query) > 0:
            search_result = Account.objects.filter(email__icontains=search_query).filter(
                username__icontains=search_query).distinct()
            user = request.user
            accounts = []
            for account in search_result:
                accounts.append((account, False))
            context['accounts'] = accounts

    return render(request, 'account/search_result.html', context)
```

Edit account View:

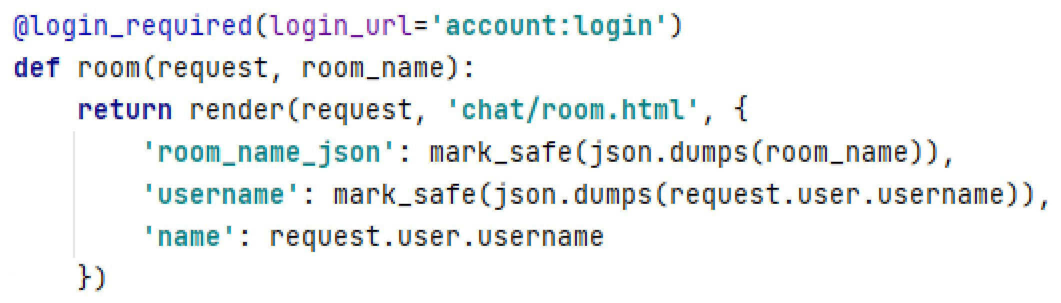


```python
@login_required(login_url='account:login')
def edit_account_view(request, *args, **kwargs):
    if not request.user.is_authenticated:
        return redirect("login")

    user_id = kwargs.get("user_id")
    try:
        account = Account.objects.get(pk=user_id)
    except Account.DoesNotExist:
        return HttpResponse("something is wrong")
    if account.pk != request.user.pk:
        return HttpResponse("you can't edit someone elses profile!!!")
    context = {}
    if request.POST:
        form = AccountUpdateForm(request.POST, instance=request.user)
        if form.is_valid():
            form.save()
            return redirect("account:profile_page", user_id=account.pk)
        else:
            form = AccountUpdateForm(request.POST, instance=request.user,
                                     initial={
                                         "id": account.pk,
                                         "email": account.email,
                                         "username": account.username
                                     }
                                     )
```



```python
            context['form'] = form
    else:
        form = AccountUpdateForm(
            initial={
                "id": account.pk,
                "email": account.email,
                "username": account.username
            }
        )
        context['form'] = form
    return render(request, "account/edit_profile.html", context)
```

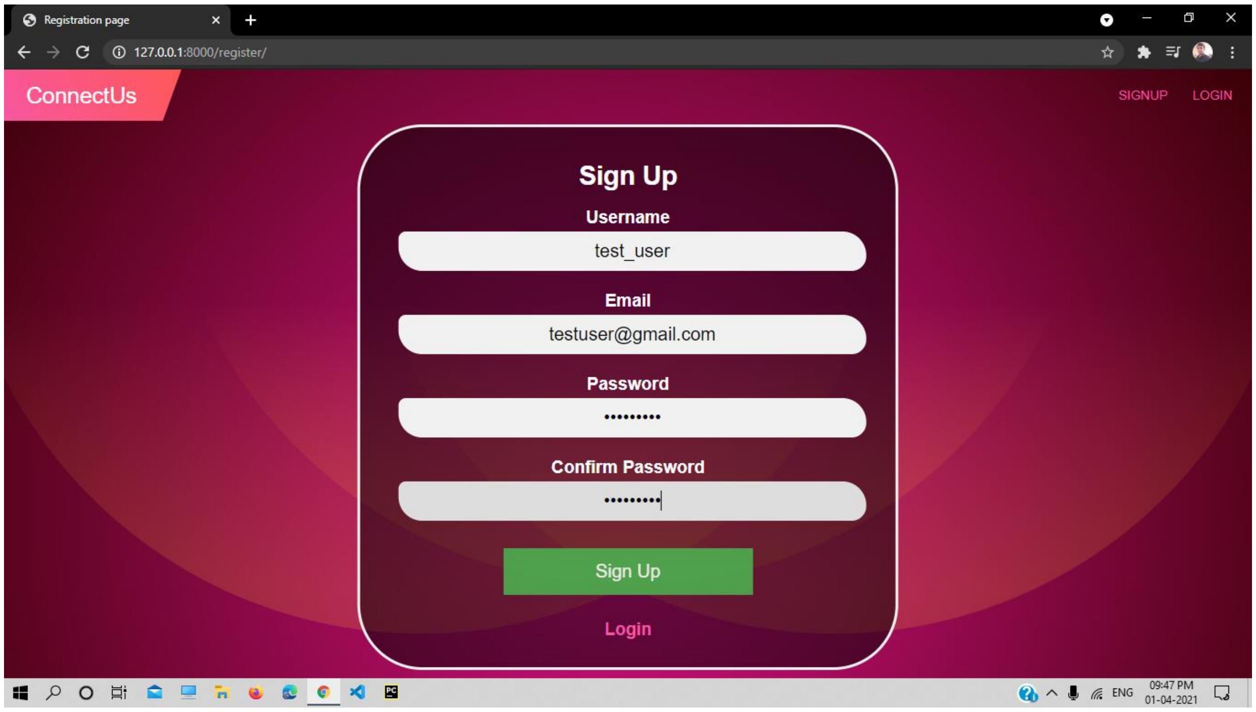
Room view:



```python
@login_required(login_url='account:login')
def room(request, room_name):
    return render(request, 'chat/room.html', {
        'room_name_json': mark_safe(json.dumps(room_name)),
        'username': mark_safe(json.dumps(request.user.username)),
        'name': request.user.username
    })
```

7. ScreenShots

signUp:



Sign Up

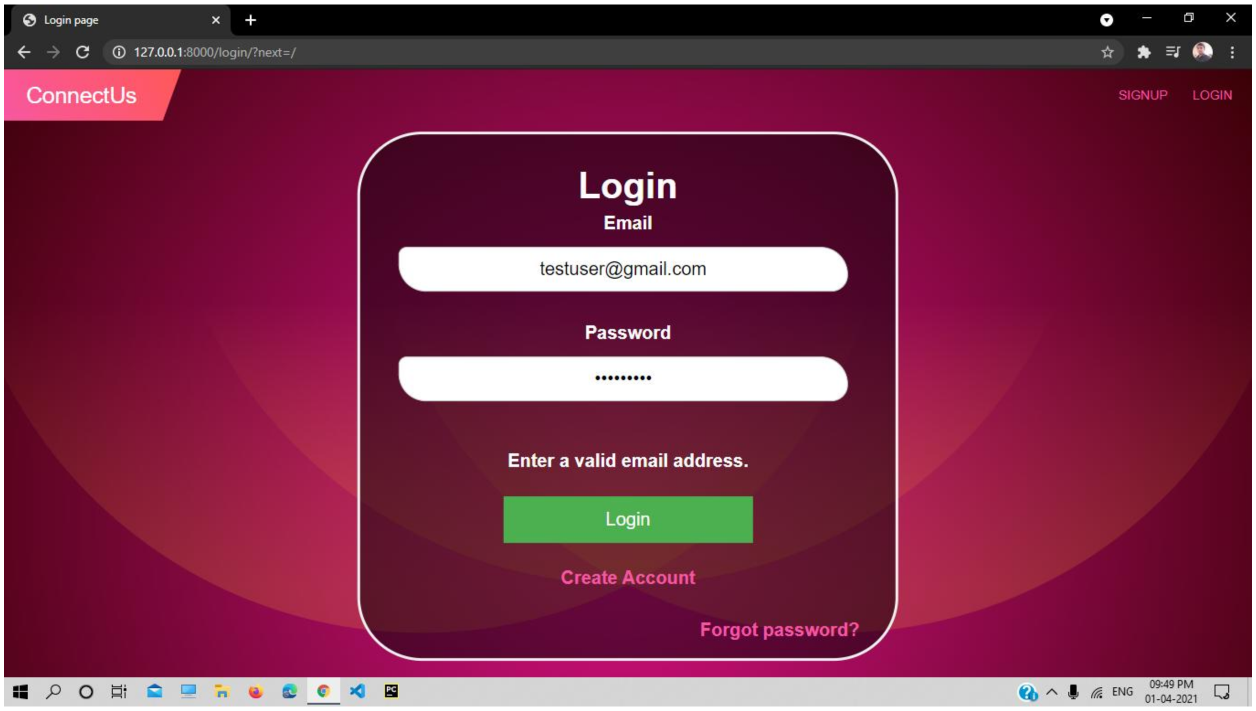
Username: test_user

Email: testuser@gmail.com

Password: •••••••••

Confirm Password: •••••••••

Login:



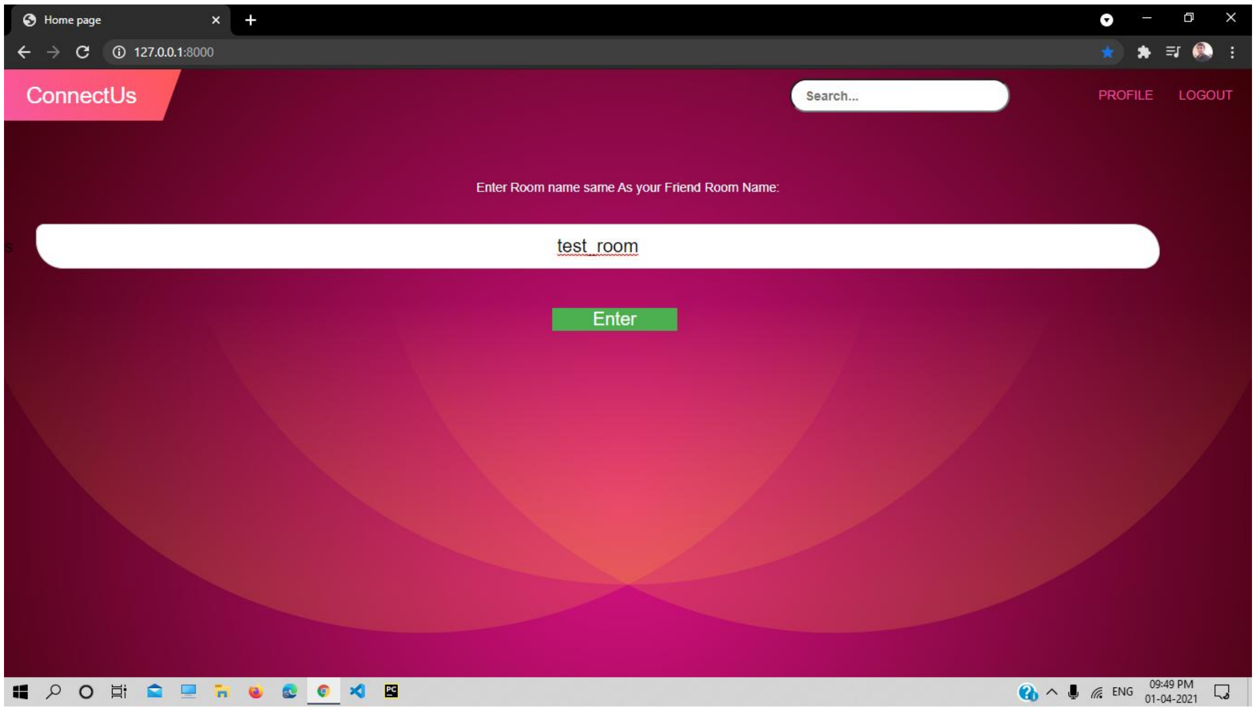
Login

Email: testuser@gmail.com

Password: •••••••••

Enter a valid email address.

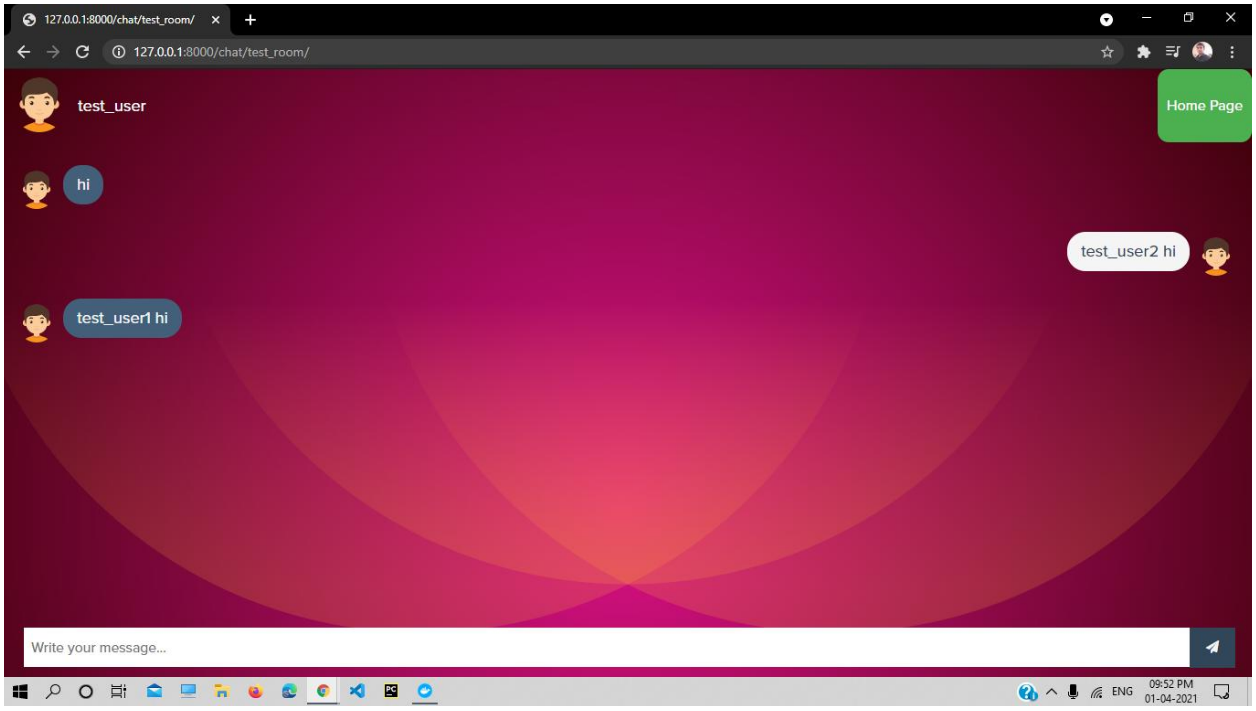
Home page:



Enter Room name same As your Friend Room Name:

test_room

User1 perspective:


127.0.0.1:8000/chat/test_room/
127.0.0.1:8000/chat/test_room/
Home Page
Write your message...
09:52 PM
01-04-2021

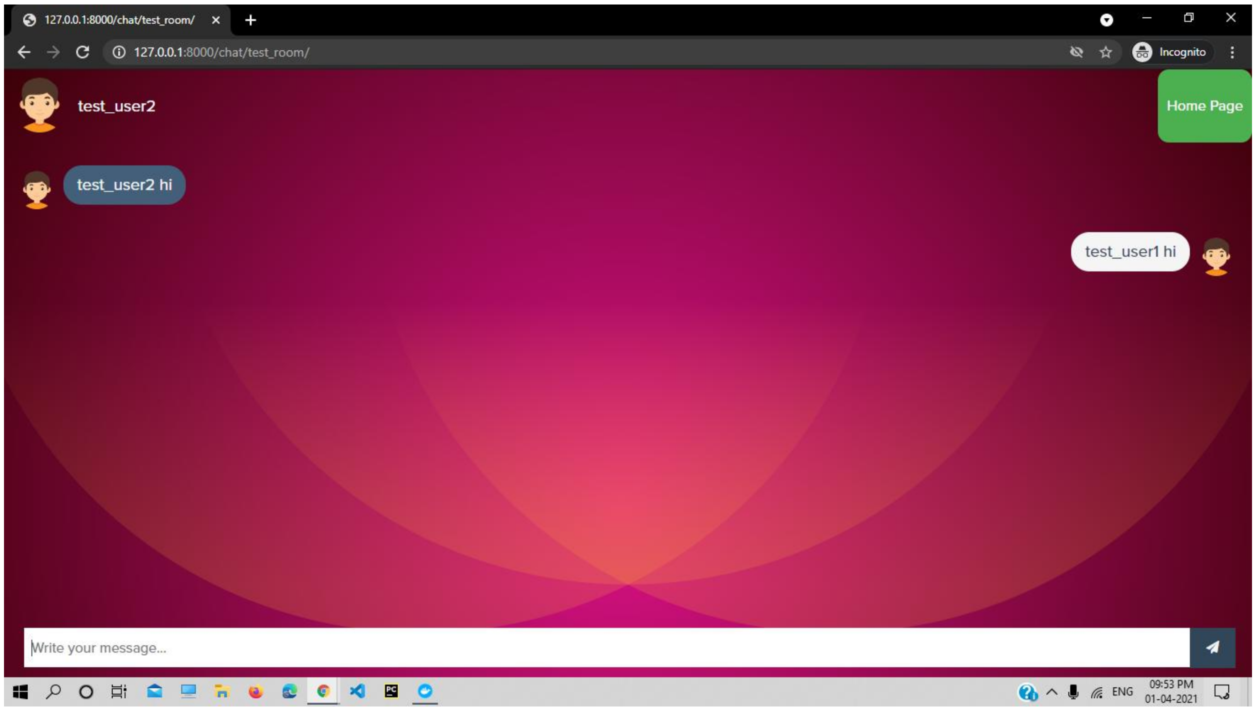
test_user

hi

test_user2 hi

test_user1 hi

User2 perspective:

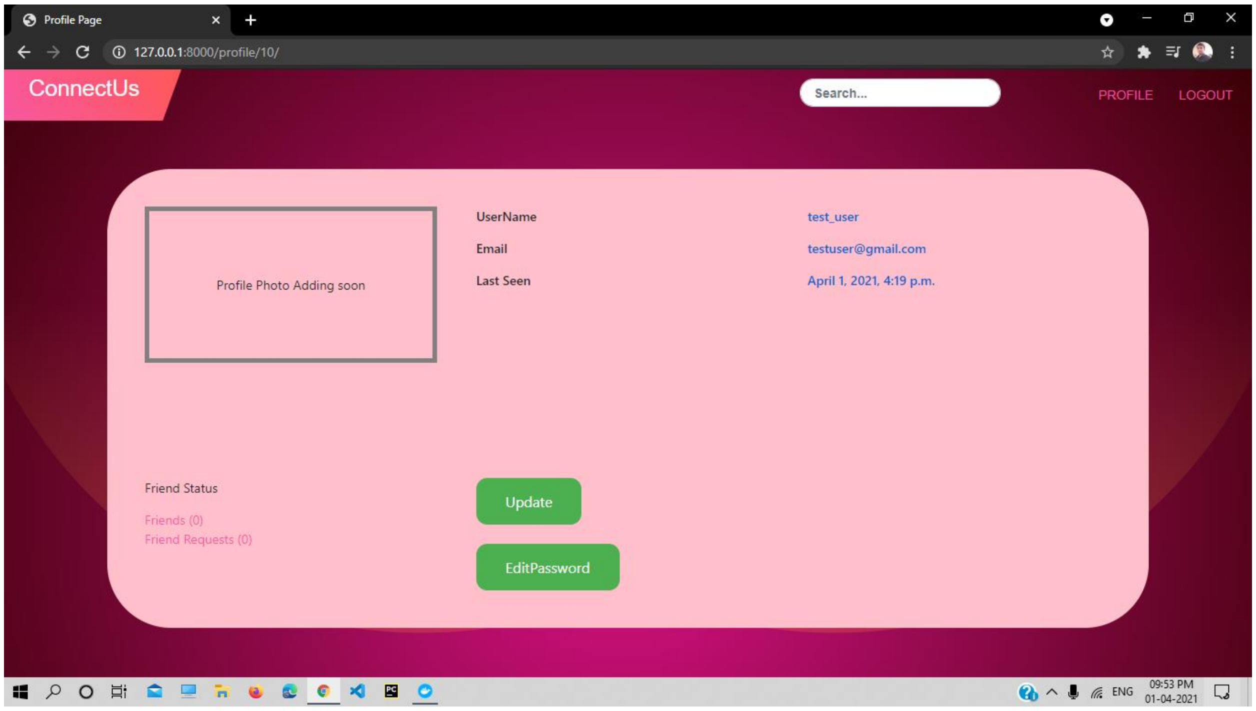


test_user2

test_user2 hi

test_user1 hi

Profile:

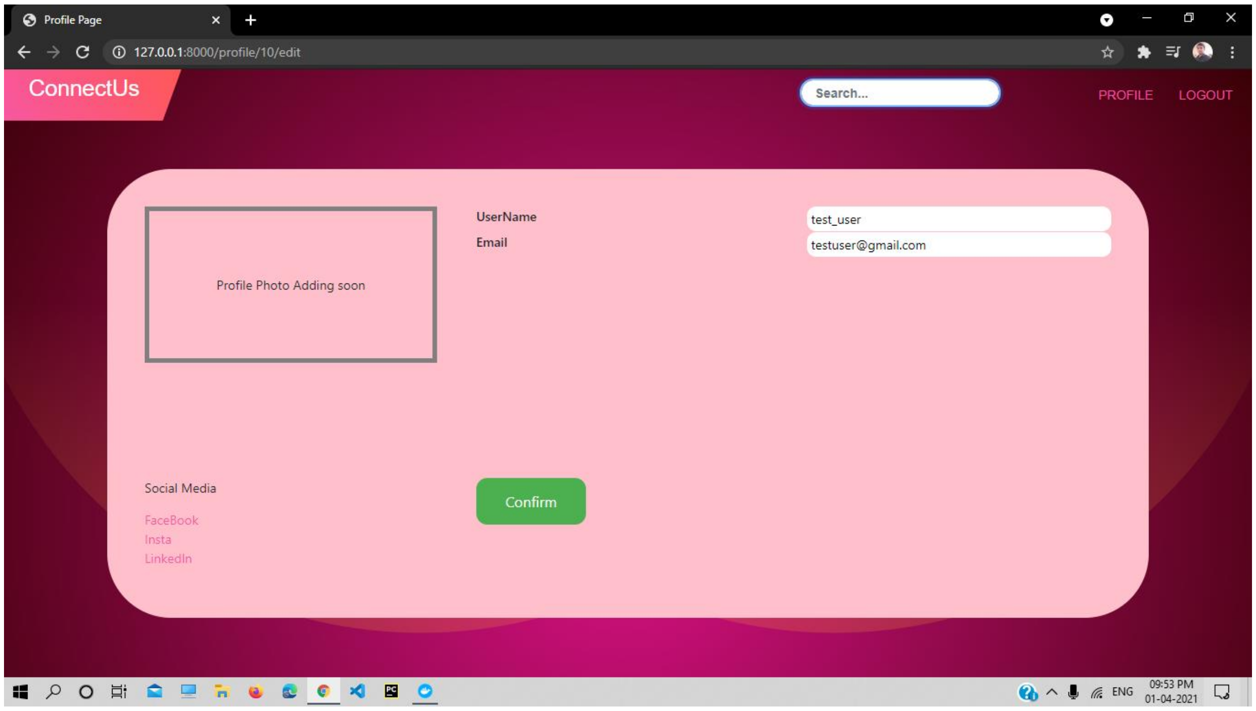


UserName	test_user
Email	testuser@gmail.com
Last Seen	April 1, 2021, 4:19 p.m.

Friend Status

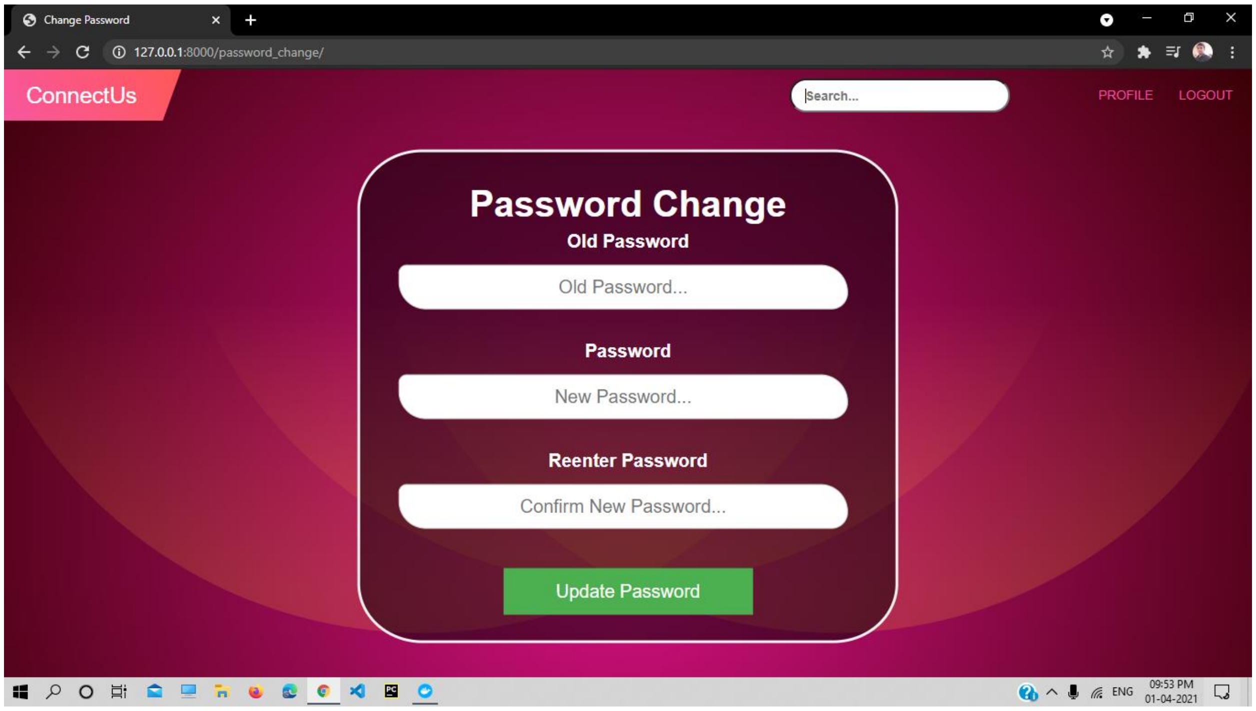
Friends (0)
Friend Requests (0)

Edit Profile:

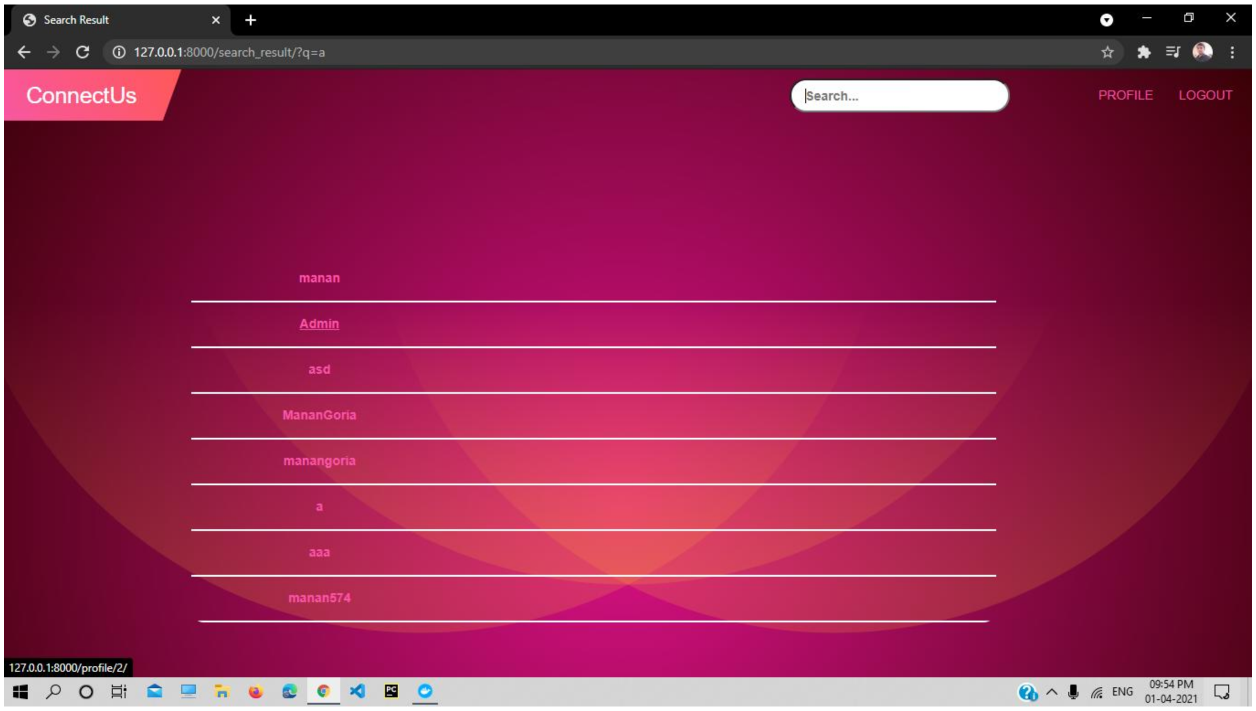


UserName test_user
Email testuser@gmail.com

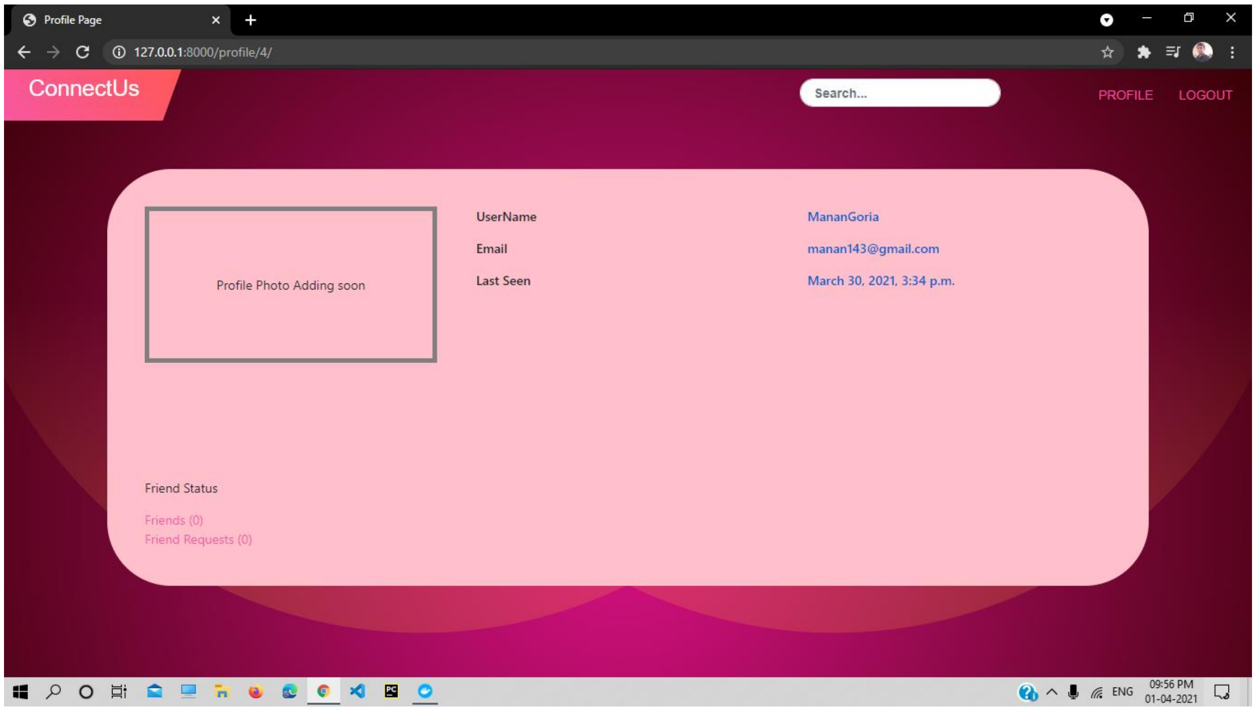
Change Password:



Search View:



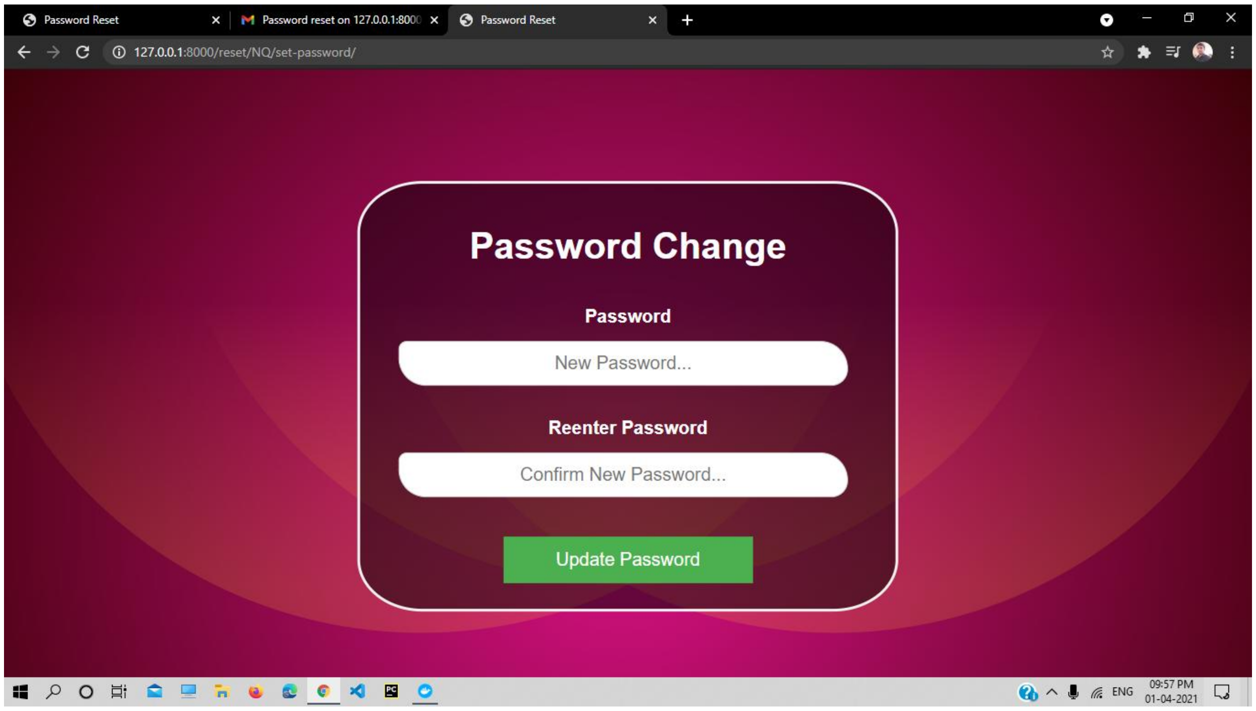
Other user's Profile:



UserName MananGoria
Email manan143@gmail.com
Last Seen March 30, 2021, 3:34 p.m.

Friend Status
Friends (0)
Friend Requests (0)

Password Reset:



7. Conclusion

The functionalities are implemented in system after understanding all the system models according to then requirements. Functionalities which are successfully implemented are:

- Login
- User authentication
- SignUp
- Logout
- Chat with other users
- Edit profile
- Reset Password

8. Limitations

1. Users can't delete their profile from database.
2. Users can't share photo in chat room.
3. If Large Amount of people joins one chat room. Server might get crash or response time will increase.

9. Future Enhancements

1. Will be adding Friend system which let user add other users as friend as well as user don't have to enter room name to chat with friends.
2. More security for Chat room which will not let other users enter chat room without permission.
3. Group chat feature.
4. Publish Web App on Global server.

10. Reference

https://docs.djangoproject.com/en/3.1/

https://channels.readthedocs.io/en/stable/

https://stackoverflow.com/

https://www.youtube.com/