به نام خدا

گزارش پروژه نهایی دوره پایتون و جنگو

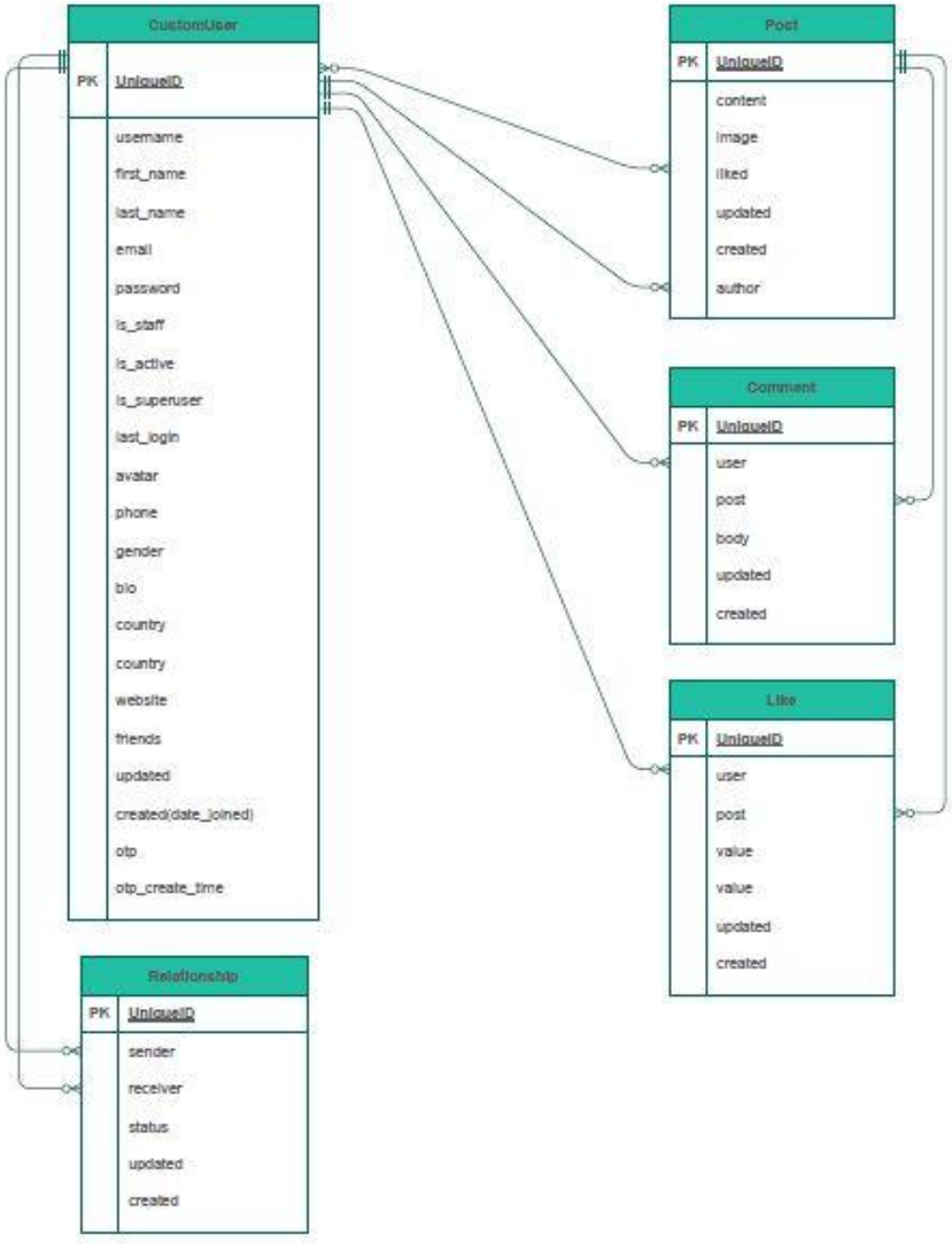
پریسا اعتمادی نژاد، سپیده نیک طبع

زمستان ۱۳۹۹

فهرست

شرح موجویت ها

در این پروژه از ۲ app شامل profiles و posts و ۵ مدل شامل post، like، comment، CustomUser و Relationship استفاده شده است. در ادامه به شرح app ها و مدل ها میپردازیم.



شکل ۱- نمودار ERD پروژه شبکه اجتماعی

Apps

در این پروژه دو app تعریف شده است.

۱. Profiles

این app برای مدیریت مشخصات کاربر و تمامی عملیات مربوط به کاربر طراحی شده است.

۲. posts

این app برای مدیریت پست هایی که کاربر ارسال می کند و تمامی عملیاتی که مربوط به پست ها است طراحی شده است.

profiles app

مدل CustomUser

این مدل معرف یک کاربر در پروژه شبکه اجتماعی می باشد. در این مدل ما مدل پیشفرض User در جنگو را override کردیم. ویژگی های این مدل به شرح زیر است:

۱- first_name
۲- last_name
۳- username
۴- avatar
۵- profile picture
۶- phone
۷- gender
۸- bio
۹- country
۱۰- validators
۱۱- website
۱۲- email
۱۳- slug
۱۴- is_active
۱۵- is_superuser
۱۶- is_staff
۱۷- friends
۱۸- updated
۱۹- created
۲۰- otp
۲۱- otp-created-time

مدل Relationship

این موجودیت رابطه بین دو کاربر شبکه اجتماعی را بیان می کند که ویژگی های این موجودیت به شرح زیر است:

۱- Sender
کاربری که درخواست دوستی را فرستاده است

۲- Receiver
کاربری که درخواست دوستی را دریافت کرده است

۳- Status
وضعیت درخواست دوستی که می تواند ارسال شده یا پذیرفته شده باشد

۴- Updated
زمان بروزرسانی درخواست

۵- Created
زمان ارسال درخواست

Authentication با شماره تلفن و ایمیل

از کلاس زیر برای ایجاد یک token برای verify با ایمیل استفاده می شود.

```
class AccountActivationTokenGenerator(PasswordResetTokenGenerator):
    """
        creates token for verifying by email
    """

    def _make_hash_value(self, user, timestamp):
        return (
                six.text_type(user.pk) + six.text_type(timestamp) +
                six.text_type(CustomUser.email_confirmed)
        )


account_activation_token = AccountActivationTokenGenerator()
```

متد زیر عملیات active کردن با ایمیل را مدیریت می کند

```python
def activate(request, uidb64, token):
    """
        function for activating user account when  user signed up  by email
    """
    try:
        uid = force_text(urlsafe_base64_decode(uidb64))
        user = CustomUser.objects.get(pk=uid)
    except(TypeError, ValueError, OverflowError, CustomUser.DoesNotExist):
        user = None
    if user is not None and account_activation_token.check_token(user, token):
        user.is_active = True
        login(request, user, backend='profiles.backends.PhoneEmailBackend')
        return redirect('home-view')
    else:
        return render(request, 'main/acc_active_email.html')
```

از متد زیر برای تولید رمز یک بار مصرف و بررسی تاریخ انقضا داشتن رمز یک بار مصرف با استفاده از کاوه نگار استفاده می شود.

```python
def get_random_otp():
    """
        creating random number for generating OTP
    """
    return randint(1000, 9999)


def check_otp_expiration(phone):
    """
        check whether OTP is expired or not
    """
    try:
        user = models.CustomUser.objects.get(phone=phone)
        now = datetime.now(timezone.utc)
        otp_time = user.otp_create_time
        diff_time = now - otp_time
        print('OTP TIME: ', diff_time)

        if diff_time.seconds > 120:
            return False
        return True

    except models.CustomUser.DoesNotExist:
        return False
```

با استفاده از متد زیر می توان backend احراز هویت در جنگو را تغییر داد

```python
class PhoneEmailBackend(ModelBackend):
    """
    Django authentication system overridden to  be able to login with Username,Email and Phone number
    """

    def authenticate(self, request, username=None, password=None, **kwargs):
        try:
            user = CustomUser.objects.get(
                Q(username__iexact=username) | Q(email__iexact=username) | Q(phone__iexact=username))
        except CustomUser.DoesNotExist:
            CustomUser().set_password(password)
        except MultipleObjectsReturned:
            return CustomUser.objects.filter(email=username).order_by('id').first()
        # except:
        #     return CustomUser.objects.filter(phone=username).order_by('id').first()
        else:
            if user.check_password(password) and self.user_can_authenticate(user):
                return user

    def get_user(self, user_id):
        try:
            user = CustomUser.objects.get(pk=user_id)
        except CustomUser.DoesNotExist:
            return None
        return user if self.user_can_authenticate(user) else None
```

از متد زیر برای ارسال رمز یک بار مصف استفاده می شود.

```python
def send_otp(phone, otp):
    """
        sending OTP to the user phone for verifying by phone
    """
    phone = [phone, ]
    try:
        api = KavenegarAPI(Kavenegar_API)
        params = {
            'sender': '1000596446',  # optional
            'receptor': phone,  # multiple mobile number, split by comma
            'message': 'Your OTP is {}'.format(otp),
        }
        response = api.sms_send(params)
        print('OTP: ', otp)
        print(response)
    except APIException as e:
        print(e)
    except HTTPException as e:
        print(e)
```

از متد زیر برای احراز هویت کاربر با استفاده از رمز یک بار مصرف استفاده می شود

```python
def verify(request):
    """
        function for activating user account when user signed up  by phone
    """
    try:
        phone = request.session.get('user_phone')
        user = CustomUser.objects.get(phone=phone)

        if request.method == "POST":

            # check otp expiration
            if not check_otp_expiration(user.phone):
                return HttpResponseRedirect(reverse('register_view'))

            if user.otp != int(request.POST.get('otp')):
                return HttpResponseRedirect(reverse('verify'))

            user.is_active = True
            user.save()
            login(request, user, backend='profiles.backends.PhoneEmailBackend')
            return HttpResponseRedirect(reverse('home-view'))

        return render(request, 'main/verify.html', {'phone': phone})

    except CustomUser.DoesNotExist:
        return HttpResponseRedirect(reverse('signup-phone'))
```

توابع view

وقتی کاربر از سمت دوست خود یک درخواست دریافت می کند دو عملکرد می تواند انجام دهد یکی اینکه درخواست را قبول کند یا اینکه آن را رد کند که در توابع زیر این عملیات مدیریت شده است.

```python
@login_required
def accept_invitation(request):
    if request.method == "POST":
        pk = request.POST.get('profile_pk')
        sender = CustomUser.objects.get(pk=pk)
        receiver = CustomUser.objects.get(id__exact=request.user.id)
        rel = get_object_or_404(Relationship, sender=sender, receiver=receiver)
        if rel.status == 'send':
            rel.status = 'accepted'
            rel.save()
    return redirect('profiles:my-invites-view')


@login_required
def reject_invitation(request):
    if request.method == "POST":
        pk = request.POST.get('profile_pk')
        receiver = CustomUser.objects.get(id__exact=request.user.id)
        sender = CustomUser.objects.get(pk=pk)
        rel = get_object_or_404(Relationship, sender=sender, receiver=receiver)
        rel.delete()
    return redirect('profiles:my-invites-view')
```

هر کاربر می تواند بعد از login لیست تمام کسانی را که امکان ارسال درخواست برای آنها وجود دارد را ببیند که این عملیات با کلاس زیر مدیریت می شود.

```python
class invite_profiles_list_view(LoginRequiredMixin, ListView):
    """profiles list available to invite"""
    model = CustomUser
    template_name = 'profiles/to_invite_list.html'
    context_object_name = 'qs'

    def get_queryset(self):
        user = self.request.user
        qs = CustomUser.objects.get_all_profiles_to_invite(user)
        return qs

    def get_context_data(self, **kwargs):
        """
        this function allows us to some additional context to the template
        """
        context = super().get_context_data(**kwargs)
        user = CustomUser.objects.get(username__iexact=self.request.user.username)  # it gets user the user
        profile = CustomUser.objects.get(id__exact=user.id)
        rel_r = Relationship.objects.filter(sender=profile)
        # relationships where we invited other users
        rel_s = Relationship.objects.filter(receiver=profile)
        # relationships where we are receiver of the invitation
        rel_receiver = []
        rel_sender = []
        for item in rel_r:
            rel_receiver.append(item.receiver.username)
```

```python
        for item in rel_s:
            rel_sender.append(item.sender.username)
        context["rel_receiver"] = rel_receiver
        context["rel_sender"] = rel_sender
        context['is_empty'] = False
        if len(self.get_queryset()) == 0:
            # if we will be the only profile 'is_empty' will be equall to True
            context['is_empty'] = True
        return context
```

هر کاربر می تواند در پروفایل خود بعد از login لیست تمام دعوت هایی که دریافت کرده است را ببیند که این عملیات با تایع زیر مدیریت می شود.

```python
@login_required
def invited_received_view(request):
    """gets all the invitations for a particular profile"""
    profile = CustomUser.objects.get(id__exact=request.user.id)
    qs = Relationship.objects.invitation_received(profile)
    results = list(map(lambda x: x.sender, qs))
    if len(results) == 0:
        is_empty = True
    else:
        is_empty = False
    context = {
        'qs': results,
        'is_empty': is_empty,
    }
    return render(request, 'profiles/my_invites.html', context)
```

هر کاربر می تواند در پروژه login کند که این عملیات با تابع زیر مدیریت می شود

```python
class LoginView(auth_views.LoginView):
    """
        View for loading logging form
    """
    form_class = LoginForm
    template_name = 'main/login.html'
```

هر کاربر می تواند در پروفایل خود بعد از login اطلاعات کاربری خود را ببیند.

```python
@login_required
def my_profile_view(request):
    """
        for displaying User profile information
    """
    profile = CustomUser.objects.get(id__exact=request.user.id)
    form = ProfileModelForm(request.POST or None, request.FILES or None, instance=profile)
    posts = profile.get_all_authors_posts()
    len_posts = True if len(profile.get_all_authors_posts()) > 0 else False
    # instance shows us which profile we want to update
    confirm = False

    if request.method == 'POST':
        if form.is_valid:
            form.save()
            confirm = True  # for update

    context = {
        'profile': profile,
        'form': form,
        'confirm': confirm,
        'posts': posts,
        'len_posts': len_posts
    }

    return render(request, 'profiles/myprofile.html', context)
```

هر کاربر می تواند برای کاربر دیگر درخواست دوستی ارسال کند و یا اینکه از سمت کاربر دیگر درخواست دوستی دریافت کند که با استفاده از کلاس زیر می توان لیستی از درخواست های دریافت شده و لیستی از درخواست ارسال شده را بدست آورد و با بررسی مقادیر این لیست ها در مورد نشان دادن دکمه های مربوطه در template تصمیم گرفت.

```python
class ProfileDetailView(LoginRequiredMixin, DetailView):
    model = CustomUser
    template_name = 'profiles/detail.html'

    def get_object(self, slug=None):
        slug = self.kwargs.get('slug')
        profile = CustomUser.objects.get(slug=slug)
        return profile

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        user = CustomUser.objects.get(username__iexact=self.request.user.username)
        profile = CustomUser.objects.get(id__exact=user.id)
        rel_r = Relationship.objects.filter(sender=profile)
        rel_s = Relationship.objects.filter(receiver=profile)
        rel_receiver = []
        rel_sender = []
        for item in rel_r:
            rel_receiver.append(item.receiver.username)
        for item in rel_s:
            rel_sender.append(item.sender.username)
        context["rel_receiver"] = rel_receiver
        context["rel_sender"] = rel_sender
        context['posts'] = self.get_object().get_all_authors_posts()
        context['len_posts'] = True if len(self.get_object().get_all_authors_posts()) > 0 else False
        return context
```

در کلاس زیر لیست همه ی کاربران با وضعیتشان نسبت به کاربر فعلی که در برنامه login است را نشان می دهد

```python
class ProfileListView(LoginRequiredMixin, ListView):
    """get all profiles by list view"""
    model = CustomUser
    template_name = 'profiles/profile_list.html'
    context_object_name = 'qs'

    def get_queryset(self):  # get all profile without own
        qs = CustomUser.objects.get_all_profiles(self.request.user)
        return qs

    def get_context_data(self, **kwargs):
        """this function allows us to some additional context to the template"""

        context = super().get_context_data(**kwargs)
        user = CustomUser.objects.get(username__iexact=self.request.user.username)  # it gets user the user
        profile = CustomUser.objects.get(id__exact=user.id)
        rel_r = Relationship.objects.filter(sender=profile)
        # relationships where we invited other users
        rel_s = Relationship.objects.filter(receiver=profile)
        # relationships where we are receiver of the invitation
        rel_receiver = []
        rel_sender = []
        for item in rel_r:
            rel_receiver.append(item.receiver.username)
        for item in rel_s:
            rel_sender.append(item.sender.username)
```

```python
            rel_sender.append(item.sender.username)
        context["rel_receiver"] = rel_receiver
        context["rel_sender"] = rel_sender
        context['is_empty'] = False
        if len(self.get_queryset()) == 0:
            # if we will be the only profile 'is_empty' will be equall to True
            context['is_empty'] = True
        return context
```

از متد زیر برای حذف یک کاربر از لیست دوستان کاربر فعلی استفاده شده است.

```python
@login_required
def remove_from_friends(request):
    """here we dont know who is the sender and who is the receiver of the request,we have to delete
     the relationship after getting it we have to remove the person who we dont want to fried with from
      our friend list and also remove ourself from that person friend list"""

    if request.method == 'POST':
        pk = request.POST.get('profile_pk')
        user = request.user
        sender = CustomUser.objects.get(id__exact=user.id)  # sender is us
        receiver = CustomUser.objects.get(pk=pk)  # profile we want to remove from friends

        # there is two senario here whether first we requested and we want to remove that guz from
        # out friends list or first that guz requested us and now we want to remove him from our friends list
        rel = Relationship.objects.get(
            (Q(sender=sender) & Q(receiver=receiver)) | (Q(sender=receiver) & Q(receiver=sender))
        )

        rel.delete()
        return redirect(request.META.get('HTTP_REFERER'))  # ??  # in order to stay on the same page
    return redirect('profiles:my-profile-view')  # if we are not dealing with post request
    # in order to get rid of user from friends list we use signals,'pre_delete signals function'
```

از متد زیر برای ارسال درخواست برای افراد موجود در این برنامه می توان استفاده کرد.

```python
@login_required
def send_invitation(request):
    """here we are the sender of invitation and we should choose the receiver,and the receiver is
    our profile pk,based on the profile pk which we get from profiles list,we get the receiver"""

    if request.method == 'POST':
        pk = request.POST.get('profile_pk')
        user = request.user
        sender = CustomUser.objects.get(id__exact=user.id)
        receiver = CustomUser.objects.get(pk=pk)  # primary key

        #  creating relationship
        rel = Relationship.objects.create(sender=sender, receiver=receiver, status='send')
        return redirect(request.META.get('HTTP_REFERER'))  # in order to stay on the same page
    return redirect('profiles:my-profile-view')  # if access to this url directly
```

متد زیر signup با استفاده از email و شماره تلفن را مدیریت می کند که اگر کاربر شماره تلفن وارد کند برای او یک پیامک حاوی یک عدد ۴ رقمی برای تایید آن ارسال می شود و اگر کاربر با ایمیل وارد شود در فایل send email پروژه یک لینک برای فرد ساخته می شود که با کلیک بر روی آن وارد صفحه کاربری خود می شود حال اگر هر دو مورد را وارد کند بصورت پیشفرض با شماره تلفن login می شود

```python
class Signup(View):
    """
        Sign up function,if user signed up by email,her/his account will be verified by sending verification email
        ,in this project email will be stored in sent_emails directory.
        if user signs up by phone a SMS will be sent to his/her phone  which contains Token number.
        if user fills both phone email filled his/her account will ber verified by sending SMS which contains
        Token number
    """

    def get(self, request):
        form = SignUpForm()
        return render(request, 'main/signup.html', {'form': form})

    def post(self, request):
        form = SignUpForm(request.POST)

        if form.is_valid():
            if form.cleaned_data['phone'] and form.cleaned_data['email']:
                phone = request.POST.get('phone')
                user = form.save(commit=False)
                # send otp
                otp = get_random_otp()
                send_otp(phone, otp)
                # save otp
                print(otp)
```

```python
                user.otp = otp
                user.is_active = False
                user.save()
                request.session['user_phone'] = user.phone
                return HttpResponseRedirect(reverse('verify'))

            elif form.cleaned_data['email']:
                user = form.save(commit=False)
                user.is_active = False
                user.save()
                current_site = get_current_site(request)
                mail_subject = 'Activate your blog account.'
                message = render_to_string('main/acc_active_email.html', {
                    'user': user,
                    'domain': current_site.domain,
                    'uid': urlsafe_base64_encode(force_bytes(user.pk)),
                    'token': account_activation_token.make_token(user),
                })
                to_email = form.cleaned_data.get('email')
                email = EmailMessage(
                    mail_subject, message, to=[to_email]
                )
                email.send()
                return HttpResponse('Please confirm your email address to complete the registration')
```

```
        elif form.cleaned_data['phone']:
            phone = request.POST.get('phone')
            user = form.save(commit=False)
            # send otp
            otp = get_random_otp()
            send_otp(phone, otp)
            # save otp
            print(otp)
            user.otp = otp
            user.is_active = False
            user.save()
            request.session['user_phone'] = user.phone
            return HttpResponseRedirect(reverse('verify'))

    return render(request, 'main/signup.html', {'form': form})
```

Autocomplete

برای بخش autocomplete پروژه از تابع آن در javascript استفاده می کنیم که بصورت زیر نوشته می شود. مقدار minlength در این تابع برابر ۱ در نظر گرفته می شود که این به معنای حداقل تایپ یک کاراکتر برای شروع جست و جو است.

```
<div class="ui icon center input centered">
    <input class="prompt" type="text" name="email" id="email" placeholder="Search friends...">
</div>

<script src="https://code.jquery.com/jquery-1.12.4.js"></script>
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>

<script>
    $(function () {
        $("#email").autocomplete({
            source: '{% url 'profiles:search-view' %}',
            minLength: 1
        });
    });
    $(document).on('click','#ui-id-1 li', function (){
        var div=$(this).find('div').text();
        window.location.href="{% url 'profiles:profile-detail-view' 0 %}".replace(0,div);
    });
</script>
```

در view مربوط به profile از تابع زیر برای بخش autocomplete پروژه استفاده می شود. در این تابع به ازای تایپ هر کاراکتر در دیتابیس می گردد و نتیجه را نشان می دهد.

```
def autocomplete(request):
    if 'term' in request.GET:
        qs = CustomUser.objects.filter(username__icontains=request.GET.get('term'))
        titles = list()
        for person in qs:
            titles.append(person.username)
        return JsonResponse(titles, safe=False)
    return render(request, 'profiles/search.html')
```

سیگنال

برای اینکه وقتی کاربر وضعیت یک relation را تغییر می دهد همان لحظه نیز در پروفایل کاربر مورد نظر بعنوان friend ثبت شود یا از لیست friend ها حذف شود، از سیگنال استفاده می شود که برای کار با سیگنال بایستی تابع ready در فایل apps باز نویسی شود و در فایلی بنام signals سیگنال های مربوطه نوشته شود.

```python
class ProfilesConfig(AppConfig):
    name = 'profiles'

    def ready(self):
        import profiles.signals
```

```python
@receiver(post_save, sender=Relationship)
def post_save_add_friends(sender, instance, created, **kwargs):
    """is used for adding friends automatically"""
    sender_ = instance.sender
    receiver_ = instance.receiver
    if instance.status == 'accepted':
        sender_.friends.add(receiver_.id)
        receiver_.friends.add(sender_.id)
        sender_.save()
        receiver_.save()


@receiver(pre_delete, sender=Relationship)  # signal is pre_delete and receiver is Relationship
def pre_delete_remove_from_friends(sender, instance, **kwargs):
    """before the relationship get deleted,it is removed from friends"""
    # here sender is instance of Relationship
    sender = instance.sender
    receiver = instance.receiver
    sender.friends.remove(receiver.id)
    receiver.friends.remove(sender.id)
    sender.save()
    receiver.save()
```

فرم

از فرم زیر برای login کاربر استفاده شده است.

```python
class LoginForm(AuthenticationForm):
    """
    Form for logging
    """
    username = forms.CharField(label='Email / Username/phone')
```

از قرم زیر برای مدیریت اطلاعات کاربر در پروفایل کاربر استفاده می شود.

```python
class ProfileModelForm(forms.ModelForm):
    """
    Form for updating user profile
    """

    class Meta:
        model = CustomUser
        fields = ('first_name', 'last_name', 'bio','gender', 'website', 'avatar')
```

```python
class SignUpForm(UserCreationForm):
    """
    form for Signup
    """

    class Meta:
        model = CustomUser
        fields = (
            'username', 'phone', 'email', 'password1', 'password2',)

    def clean(self):
        cleaned_data = super().clean()
        if cleaned_data.get('email') is None and cleaned_data.get('phone') is None:
            raise ValidationError('Please Enter an Email or a Phone number')
        return cleaned_data
```

Context processor

برای ارسال مقادیر پویا به template می توان از context processor بصورت زیر استفاده کرد.

```python
def profile_pic(request):
    """
    for Sending Profile picture which is a dynamic content to the template
    """
    if request.user.is_authenticated:
        profile_obj = CustomUser.objects.get(id__exact=request.user.id)
        pic = profile_obj.avatar
        return {'picture': pic}
    return {}


def invitation_received_no(request):
    """
    for Sending number of Invitations which is a dynamic content to the template
    """
    if request.user.is_authenticated:
        profile_obj = CustomUser.objects.get(id__exact=request.user.id)
        qs_count = Relationship.objects.invitation_received(profile_obj).count()
        return {'invites_num': qs_count}
    return {}
```

Posts app

مدل post:

این موجودیت مشخصات یک پست را بیان می کند. ویژگی های این موجودیت شامل موارد زیر است:

۱- Content
۲- Image
۳- Liked
۴- Updated
۵- Created

مدل like :

این موجودیت like هر پست را مشخص می کند. ویژگی های این موجودیت شامل موارد زیر است.

۱- User
کاربری که لایک کرده مشخص می شود
۲- Post

پستی که لایک شده مشخص می شود

۳- Updated

زمان بروزرسانی لایک

۴- Created

زمان انجام لایک

.

مدل comment :

این موجودیت کامنتی که کاربر زیر هر پست درج می کند را مشخص می کند.

۱- User

کاربری که کامنت را درج کرده است

۲- Post

پستی که کامنت زیر آن درج شده است

۳- Body

محتوایی که کامنت شده است

۴- Updated

زمان بروزرسانی کامنت

۵- Created

زمان درج کامنت

توابع view

از کلاس زیر برای حذف کامنت یک پست استفاده می شود.

```
class CommentDeleteView(LoginRequiredMixin, DeleteView):
    """
       the function increase security and prevents deleting of the post by other users who knows url of a page
       and only the author can delete the post
    """
    model = Comment
    template_name = 'posts/confirm_del.html'
    success_url = reverse_lazy('posts:main-post-view')  # redirects to the main-post-view

    def get_object(self, *args, **kwargs):
        pk = self.kwargs.get('pk')
        obj = Comment.objects.get(pk=pk)
        return obj
```

متد زیر like و unlike کردن یک پست را مدیریت می کند.

```python
@login_required
def like_unlike_post(request):
    """
    method for managing like and unlike posts
    """
    user = request.user  # user that is logged in
    if request.method == 'POST':
        post_id = request.POST.get('post_id')
        post_obj = Post.objects.get(id=post_id)
        profile = CustomUser.objects.get(id__exact=user.id)
        # check if profile of the user is in many to many field(like field in Post)

        if profile in post_obj.liked.all():
            # because the profile is already in the liked.all(),if the profile
            # liked the post again we should remove  the profile
            post_obj.liked.remove(profile)
        else:
            post_obj.liked.add(profile)

        like, created = Like.objects.get_or_create(user=profile, post_id=post_id)
        # user is the foreign key to profile
        # if created is equal to True this means  that the post didnt exist before it created
        if not created:
            if like.value == 'Like':
                like.value = 'Unlike'
            else:
```

```python
        else:
            like.value = 'Like'

            post_obj.save()
            like.save()

    return redirect('posts:main-post-view')
```

از متد زیر برای حذف یک پست تنها توسط نویسنده پست استفاده می شود.

```python
class PostDeleteView(LoginRequiredMixin, DeleteView):
    """the function increase security and prevents deleting of the post by other users who knows url of a page
       and only the author can delete the post"""
    model = Post
    template_name = 'posts/confirm_del.html'
    success_url = reverse_lazy('posts:main-post-view')  # redirects to the main-post-view

    # success_url = '/posts/'
    # it indicates once we successfully delete a post where should we be taken
    # reverse is for function views and reverse_lazy is for class views

    def get_object(self, *args, **kwargs):
        pk = self.kwargs.get('pk')
        obj = Post.objects.get(pk=pk)
        if not obj.author.id == self.request.user.id:
            messages.warning(self.request, 'You need to be the author of the post in order to delete it')
        return obj
```

از کلاس زیر برای بروزرسانی یک پست استفاده می شود

```python
class PostUpdateView(LoginRequiredMixin, UpdateView):
    """
    Updating posts
    """
    form_class = PostModelForm
    model = Post
    template_name = 'posts/update.html'
    success_url = reverse_lazy('posts:main-post-view')

    def form_valid(self, form):
        profile = CustomUser.objects.get(id__exact=self.request.user.id)
        if form.instance.author == profile:
            return super().form_valid(form)
        else:
            form.add_error(None, "You need to be the author of the post in order to update it")
            return super().form_invalid(form)
```

فرم

از فرم زیر برای ثبت کامنت استفاده می شود.

```python
class CommentModelForm(forms.ModelForm):
    """
        this form is for adding Comment for posts
    """
    body = forms.CharField(label='', widget=forms.TextInput(attrs={'placeholder': 'Add a comment...'}))

    class Meta:
        model = Comment
        fields = ('body',)
```

از فرم زیر برای ثبت پست جدید استفاده می شود.

```python
class PostModelForm(forms.ModelForm):
    """
        content overridden because we wanted our form have just 2 row,
        this form is for adding posts by which user cant add post image and content post
    """

    content = forms.CharField(widget=forms.Textarea(attrs={'rows': 2}))

    class Meta:
        model = Post
        fields = ('content', 'image',)
```

کتابخانه ها:

از کتابخانه های زیر در پروژه علاوه بر کتابخانه های معمول استفاده شده است.

کتابخانه pillow:

از این کتابخانه برای مدیریت عکس ها و دریافت عکس از کاربر استفاده می شود.

کتابخانه six:

از این کتابخانه برای بخش authentication در قسمت تاییدیه با ایمیل استفاده می شود.

```python
class AccountActivationTokenGenerator(PasswordResetTokenGenerator):
    """
        creates token for verifying by email
    """

    def _make_hash_value(self, user, timestamp):
        return (
                six.text_type(user.pk) + six.text_type(timestamp) +
                six.text_type(CustomUser.email_confirmed)
        )


account_activation_token = AccountActivationTokenGenerator()
```

کتابخانه kavehnegr:

از این کتابخانه برای ارسال پیامک حاوی عددی ۴ رقمی برای تاییدیه ثبت نام استفاده شده است.