بسم الله الرحمن الرحیم

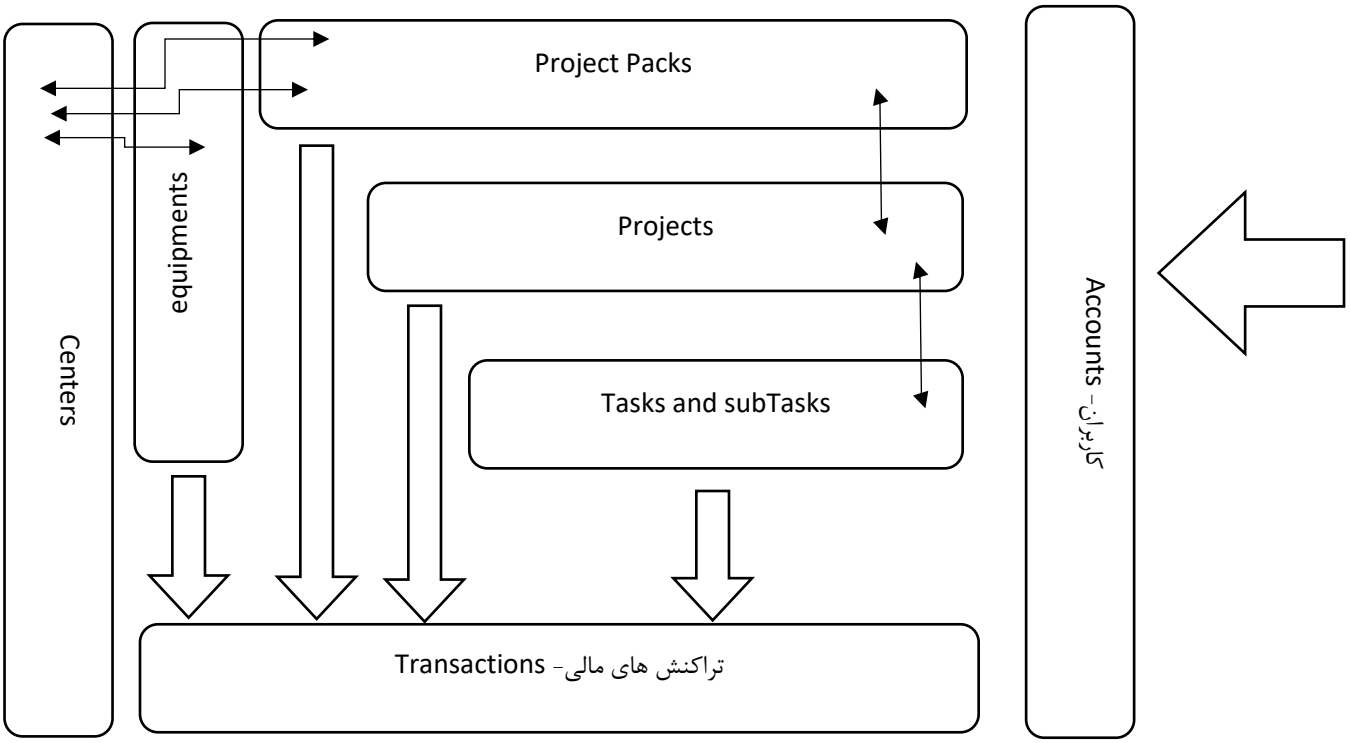
سامانه جامع مدیریت مراکز CMS

سند پیاده سازی کد

منطق کلی

سامانه cms که بکند آن با استفاده از جنگو پیاده سازی شده است دارای منطق ساده ای است. به صورت کلی و بر اساس منطق شی گرایی تمام موجودیت های قابل استفاده در سامانه به صورت کلاس(مدل ها)یی پیاده سازی شده اند و به این ترتیب از طریق لایه زیرین جنگو اشیاء ساخته از کلاس ها به صورت رکوردهای مختلف در دیتابیس ذخیره می شوند. وظیفه ارتباط با دیتابیس سامانه که postgres است نیز به عهده خود جنگو میباشد و ارتباط با رکوردها نیز از طریق مدل ها و اشیاء آن است و ارتباط مستقیمی با تیبل های داده ها نداریم.

به صورت سامانه ای هر سیستم کلی نیاز به یک ساختار هویت سنجی و پنل ورود و خروج دارد. در cms با استفاده از اپ accounts این مسئله مدیریت می شود. از طرف دیگر از طریق اپ centers موجودیت مرکز، پک پروژه، پروژه، تسک ها و زیرتسک ها تعریف و کنترل می شوند. در اپ equipments نیز تجهیزات مختلف که متعلق به مراکز و مورد استفاده در پروژه ها هستند مدیریت می شوند. در نهایت نیز فرآیندهای مالی پروژه ها توسط اپ transactions کنترل میگردد. شکل زیر شمای کلی از اپ های در ارتباط با یکدیگر را نشان میدهد.



اپ های کوچکتر

در این سند سعی شده تا به صورت دقیق و جزئی پیاده سازی هر کدام از اپ ها که کنترل کننده بخشی از سامانه است به صورت کامل تشریح گردد. به این ترتیب در صورت نیاز به توسعه سامانه نیروی برنامه نویس ماهر با مطالعه این سند به راحتی می تواند بخش های مختلف کد را اصلاح کند و یا تغییر دهد.

هر اپ از پنج بخش کلی مدل ها، نماها، فرم ها، لینک ها و تمپلیت ها تشکیل شده است که برای هر کدام علاوه بر توضیحات مفصل نحوه کارکرد اپ، محتویات دو بخش مهمتر مدلها و نماها به صورت جداگانه تشریح خواهد شد. بخش تمپلیت ها که به عنوان ظاهر و فرانت اند کار در نظر گرفته می شود از ترکیب template های jinja که زبان ارتباط بکند و فرانتند در پلتفرم جنگوست و تگ های html و css در قالب bootstrap ٤ تشکیل شده است. محتویاتی که قرار است در تمپلیت ها نمایش داده شوند؛ ابتدا توسط کوئری هایی در بخش نماها صدا زنده می شوند و سپس از آنجا عموما در دیکشنری context به تمپلیت ها پاس داده می شوند تا در آنجا لابلای تگ های html رندر شوند.

قسمت url هر اپ نیز تعیین کنند لینکی است که پل ارتباطی بین صفحه وب(همان تمپلیت مورد نظر) و نمای متناظر با آن است. در بخش لینک ها، به هر نما یک لینک مشخص با یک آدرس مشخص اختصاص داده می شود تا در صورت وارد شدن در مرورگر به سراغ آن تمپلیت و در نتیجه نمای متناظرش برود و اطلاعات را پس از استخراج از دیتابیس رندر کند.

accounts

models

در اینجا اول از همه اطلاعات پیشینی مورد نیاز مدل ها به صورت لیستی از تاپلها تعریف شده اند. دانشگاه ها، جنسیت، مدارک، زبان های برنامه نویسی، نرم افزارهای مسلط، رشته های دانشگاهی و ... اطلاعات اولیه ای هستند که در قالب توانایی های کاربر در سامانه ذخیره می شوند.

```
UNINVERSITIES = [
    ('Sharif', 'دانشگاه صنعتی شریف'),
    ('Tehran', 'دانشگاه تهران'),
    ('Elmos', 'دانشگاه علم و صنعت ایران'),
    ('Amirkabir', 'دانشگاه صنعتی امیرکبیر'),
    ('Beheshti', 'دانشگاه شهید بهشتی'),
    ('Public', 'دانشگاه سراسری'),
    ('Azad', 'دانشگاه آزاد'),
    ('Pyam', 'دانشگاه پیام نور'),
    ('Privte', 'دانشگاه غیر انتفاعی'),
]

SEXES = [
    ('man', 'مرد'),
    ('woman', 'زن'),
]

DEGREES = [
    ('diploma', 'دیپلم'),
    ('bachelor', 'کارشناسی'),
    ('master', 'ارشد'),
    ('phd', 'دکتری'),
]
```

```
LANGS = [
    ('C', 'C'),
    ('Python', 'Python'),
    ('Java', 'Java'),
    ('C++', 'C++'),
    ('Django', 'Django'),
    ('Laravel', 'Laravel'),
    ('GO', 'GO'),
]

SOFTWARES = [
    ('cat', 'اتوکد'),
    ('photoshop', 'فتوشاپ'),
    ('corel', 'کرل'),
    ('indesign', 'ایندیزاین'),
    ('catia', 'کتیا'),
]

MEASURES = [
    ('software', 'نرم افزار'),
    ('electronic', 'الکترونیک'),
    ('mechanic', 'مکانیک'),
    ('control', 'کنترل'),
    ('civil', 'عمران'),
]
```

اپ اکانت وظیفه نگه داشتن اطلاعات کاربران و authenticate کردن آن ها هنگام ورود و تعیین سطح دسترسی و نقش کاربر را دارد. بنابر این دسترسی های مورد استفاده در سامانه بر اساس نقش های مورد نیاز طراحی شده است و با استفاده از آن ها هنگام render کردن template هر صفحه متناسب با نقش خاص render می شود.

```
PERMISSIONS = [
    ('users', 'مدیریت کاربران'),
    ('centers', 'مدیریت مراکز'),
    ('projectpacks', 'مدیریت پک پروژه ها'),
    ('projects', 'مدیریت پروژه ها'),
    ('equipments', 'مدیریت تجهیزات'),
    ('transactions', 'مدیریت تراکنش ها'),
    #
    ('center', 'مدیریت مرکز'),
    ('user', 'مدیریت کاربران مرکز'),
    ('equipment', 'مدیریت تجهیزات مرکز'),
    ('transaction', 'مدیریت تراکنش های مرکز'),
    ('projectpack', 'مدیریت پک پروژه های مرکز'),
    ('projectpack_monitoring', 'کارشناس کنترل پروژه های مرکز'),
    ('project', 'پیمانکار پروژه های مرکز'),
    ('project_monitoring', 'نظارت پروژه های مرکز'),
]
```

در ادامه دو تابع تعریف شده اند که رزومه و آواتار کاربران را در یک آدرس مشخص- که بر اساس تاریخ است- ذخیره می کنند. این توابع هنگام ایجاد شدن آواتار و ساخته شدن رزومه صدا زده می شوند و مسیر مورد نظر را در سرور ایجاد می کنند.

```
def get_avatar_directory_path(instance, filename):
    return 'users/%d_avatar_%s' % (int(time.time()), filename)


def get_resume_directory_path(instance, filename):
```

```
    return 'users/%d_resume_%s' % (int(time.time()), filename)
```

و در نهایت به بخش کلاسها و یا همان مدل های مورد استفاده در این اپ میرسیم. مدل های Lang و Software و Permission به عنوان مدل های میانی جهت ذخیره سازی اطلاعات اصلی کاربر در مدل Profile استفاده می شوند. مدل Profile به عنوان کلاس اصلی این اپ تمام اطلاعات مورد نیاز را ذخیره سازی میکند.

```
class Lang(models.Model):
    name = models.CharField(max_length=۲۰, choices=LANGS, default='Python',
verbose_name='زبان برنامه نویسی')

    def __str__(self):
        return self.name


class Software(models.Model):
    name = models.CharField(max_length=۲۰, choices=SOFTWARES, default='photoshop',
verbose_name='نرم افزارهای تخصصی')

    def __str__(self):
        return self.name


class Permission(models.Model):
    code = models.CharField(max_length=۳۰, choices=PERMISSIONS, verbose_name='کد')

    def __str__(self):
        return self.code


class Profile(models.Model):
    user = models.OneToOneField(to=User, on_delete=models.CASCADE,
verbose_name='کاربر')
    avatar = models.ImageField(upload_to=get_avatar_directory_path,
verbose_name='Avatar', max_length=۲۵۵, blank=True,
                               null=True)
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='تاریخ ایجاد')
    permissions = models.ManyToManyField(to=Permission, blank=True,
verbose_name='دسترسی ها')

    mobile_number = models.CharField(max_length=۱۱, verbose_name='تلفن همراه',
unique=True)
    phone_number = models.CharField(max_length=۲۰, default='', blank=True,
verbose_name='تلفن ثابت')
    address = models.TextField(default='', blank=True, verbose_name='نشانی')

    sex = models.CharField(max_length=۲۰, blank=True, choices=SEXES, default='man',
verbose_name='جنسیت')
    degree = models.CharField(max_length=۲۰, blank=True, choices=DEGREES,
default='bachelor', verbose_name='مدرک')
    measure = models.CharField(max_length=۲۰, blank=True, choices=MEASURES,
default='software', verbose_name='رشته')
    university = models.CharField(max_length=۲۰, blank=True, choices=UNINVERSITIES,
default='Public', verbose_name='دانشگاه')
    langs = models.ManyToManyField(to=Lang, blank=True, verbose_name=' زبان های برنامه
نویسی')
    softwares = models.ManyToManyField(to=Software, blank=True, verbose_name=' نرم
افزارهای مسلط')
    other_abilities = models.TextField(default='', blank=True, verbose_name=' سایر
توانایی ها')
```

```
    resume = models.FileField(upload_to=get_resume_directory_path,
verbose_name='رزومه', max_length=۲۰۰, blank=True,
                              null=True)

    class Meta:
        ordering = ['user__last_name', 'user__first_name']

    def __str__(self):
        try:
            return self.user.get_full_name()
        except:
            return f'profile {self.id}'

    @property
    def name(self):
        return self.user.get_full_name()

    @property
    def get_user_center(self):
        center = self.center_set.first()
        return center.id

    @property
    def get_user_center_display(self):
        center = self.center_set.first()
        return center.name

    @property
    def get_user_main_role_display(self):
        main_role = self.permissions.all().first()
        if main_role:
            return main_role.get_code_display()
        return 'عضو مرکز'

    @property
    def has_no_perms(self):
        if self.permissions.all().count():
            return False
        return True

    def has_perms(self, perms):
        if self.permissions.filter(code__in=perms).count():
            return True
        return False

    def get_absolute_url(self):
        return reverse('accounts:profile_details', kwargs={'pk': self.id})
```

دو تابع انتهایی نیز برای حذف آواتار و رزومه کاربر استفاده می شوند که به عنوانی سیگنالی بر روی مدل Profile تعریف شده اند. سیگنال pre_save که قبل از ذخیره سازی شی و سیگنال pre_delete که قبل از حذف رکورد فراخوانی می شود.

```
@receiver(pre_save, sender=Profile,
dispatch_uid="delete_old_avatar_image_before_save")
def delete_old_avatar_image(sender, instance, raw, update_fields, **kwargs):
    try:
        obj = Profile.objects.get(id=instance.id)
        if obj.avatar != instance.avatar:
            obj.avatar.delete(save=False)
    except Exception:
```

```
        pass


@receiver(pre_delete, sender=Profile,
dispatch_uid='delete_avatar_file_on_object_delete')
def delete_avatar_media(sender, instance, using, **kwargs):
    instance.avatar.delete(save=False)
```

views

نماهای کلی این اپ محدود به چهار نوع کلی ساخت و حذف و اصلاح و لیست کردن اشیا ساخته شده از مدل هاست. به گونه ای می توان آن ها را معادل همان رکوردهای دیتابیس گرفت. برخی اطلاعات به صورت دیفالت به تمپلیت ها پاس داده می شوند، برخی از آن ها نیز باید به صورت جداگانه از دیتابیس فراخوانی شوند؛ هنگامی که کوئری ها صدا زده شدند جهت ارسال آن ها به تمپلیت باید از طریق دیکشنری kwargs اقدام کرد.

```
class ProfileListView(LoginRequiredMixin, RoleMixin, PaginatedFilteredListView):
    model = Profile
    template_name = 'accounts/profile_list.html'


class CenterProfileListView(LoginRequiredMixin, RoleMixin, PaginatedFilteredListView):
    model = Profile
    template_name = 'accounts/profile_list.html'

    def get_queryset(self):
        center =
Center.objects.get(id=self.request.resolver_match.kwargs['center_pk'])
        return center.employees.all()

    def get_context_data(self, **kwargs):
        kwargs['in_center'] = True
        kwargs['the_center'] =
Center.objects.get(id=self.request.resolver_match.kwargs['center_pk'])
        return super().get_context_data(**kwargs)


class ProjectProfileListView(LoginRequiredMixin, RoleMixin,
PaginatedFilteredListView):
    model = Profile
    template_name = 'accounts/profile_list.html'

    def get_queryset(self):
        project =
Project.objects.get(id=self.request.resolver_match.kwargs['project_pk'])
        return project.employees.all()


class ProfileDetailsView(LoginRequiredMixin, RoleMixin, DetailView):
    model = Profile
    template_name = 'accounts/profile_detail.html'



class CenterProfileCreateView(LoginRequiredMixin, RoleMixin, CreateView):
    model = Profile
    form_class = ProfileCreateForm
    template_name = 'accounts/profile_form.html'

    def get_success_url(self):
```

```
        return reverse_lazy('accounts:center_profile_list', kwargs={'center_pk':
self.object.get_user_center})


    def get_context_data(self, **kwargs):
        kwargs['tittle'] = 'افزودن'
        kwargs['centers'] =
Center.objects.filter(id=self.request.resolver_match.kwargs['center_pk'])
        kwargs['in_center'] = True
        kwargs['the_center'] =
Center.objects.get(id=self.request.resolver_match.kwargs['center_pk'])
        kwargs['roles'] = PERMISSIONS
        kwargs['sexes'] = SEXES
        kwargs['degress'] = DEGREES
        kwargs['measures'] = MEASURES
        kwargs['universities'] = UNINVERSITIES
        kwargs['langs'] = Lang.objects.all()
        kwargs['softwares'] = Software.objects.all()
        kwargs['permissions'] = Permission.objects.all()
        return super().get_context_data(**kwargs)


class ProfileCreateView(LoginRequiredMixin, RoleMixin, CreateView):
    model = Profile
    form_class = ProfileCreateForm
    template_name = 'accounts/profile_form.html'
    success_url = reverse_lazy('accounts:profile_list')

    def get_context_data(self, **kwargs):
        kwargs['tittle'] = 'افزودن'
        kwargs['centers'] = Center.objects.all()
        kwargs['roles'] = PERMISSIONS
        kwargs['sexes'] = SEXES
        kwargs['degress'] = DEGREES
        kwargs['measures'] = MEASURES
        kwargs['universities'] = UNINVERSITIES
        kwargs['langs'] = Lang.objects.all()
        kwargs['softwares'] = Software.objects.all()
        kwargs['permissions'] = Permission.objects.all()
        return super().get_context_data(**kwargs)


class CenterProfileUpdateForm(LoginRequiredMixin, RoleMixin, UpdateView):
    model = Profile
    form_class = ProfileUpdateForm
    template_name = 'accounts/profile_form.html'

    def get_success_url(self):
        return reverse_lazy('accounts:center_profile_list', kwargs={'center_pk':
self.object.get_user_center})

    def get_context_data(self, **kwargs):
        kwargs['tittle'] = 'ویرایش'
        kwargs['centers'] =
Center.objects.filter(id=self.request.resolver_match.kwargs['center_pk'])
        kwargs['in_center'] = True
        kwargs['the_center'] =
Center.objects.get(id=self.request.resolver_match.kwargs['center_pk'])
        kwargs['roles'] = PERMISSIONS
        kwargs['sexes'] = SEXES
        kwargs['degress'] = DEGREES
        kwargs['measures'] = MEASURES
        kwargs['universities'] = UNINVERSITIES
```

```
        kwargs['langs'] = Lang.objects.all()
        kwargs['softwares'] = Software.objects.all()
        kwargs['permissions'] = Permission.objects.all()
        return super().get_context_data(**kwargs)


class ProfileUpdateForm(LoginRequiredMixin, RoleMixin, UpdateView):
    model = Profile
    form_class = ProfileUpdateForm
    template_name = 'accounts/profile_form.html'
    success_url = reverse_lazy('accounts:profile_list')

    def get_context_data(self, **kwargs):
        kwargs['tittle'] = 'ویرایش'
        kwargs['centers'] = Center.objects.all()
        kwargs['roles'] = PERMISSIONS
        kwargs['sexes'] = SEXES
        kwargs['degress'] = DEGREES
        kwargs['measures'] = MEASURES
        kwargs['universities'] = UNINVERSITIES
        kwargs['langs'] = Lang.objects.all()
        kwargs['softwares'] = Software.objects.all()
        kwargs['permissions'] = Permission.objects.all()
        return super().get_context_data(**kwargs)


class ProfileDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = Profile

    def delete(self, request, *args, **kwargs):
        self.object = self.get_object()
        self.object.user.delete()
        self.object.delete()
        return self.render_to_response(self.get_context_data(success=True))
```

هر کدام از این نماها علاوه بر اختصاص به مدل profile به یک تمپلیت خاص نیز متناظر است که رکوردهای گرفته شده از دیتابیس توسط کوئری های مختلف در آنجا رندر می شوند و به نوعی فرانت اند سامانه را می سازند.

centers

models

در این اپ موجودیت های مرکز و پک پروژه و پروژه تعریف شده اند. هر کدام از مدل های پک پروژه و پروژه و تسک و زیرتسک یک فیلد status دارند که وضعیت شی را در لحظه مشخص می کند. هر کدام از این وضعیت ها توسط سیگنال هایی به صورت اتوماتیک فراخوانی شده و تغییر پیدا میکنند.

```
STATUS_TYPES = [
    ('to_do', 'جهت انجام'),
    ('inprogress', 'در حال انجام'),
    ('completed', 'انجام شده'),
    ('verified', 'تایید شده'),
]

TRANSACTION_TITTLES = [
    ('prepayment', 'پیش پرداخت'),
    ('installment', 'قسط'),
    ('paragraph', 'بند'),
```

```
]
```

هر مرکز علاوه بر نگهداری اطلاعات ایجاد کننده یک رابطه foreign key با profile به عنوان مدیر مرکز و یک رابطه ManytoMany با profile به عنوان لیست کارکنان مرکز دارد. به دین ترتیب به صورت کلی ارتباطات کلی مرکز و کارمندان از این طریق حفظ و نگهداری می شود.

```
class Center(models.Model):
    name = models.CharField(max_length=۲۵۵, verbose_name='نـام', unique=True)
    description = models.TextField(null=True, blank=True, verbose_name='تـوضیحات')

    created_at = models.DateTimeField(auto_now_add=True, verbose_name='تـاریـخ ایـجاد')
    # created_by = models.ForeignKey(to=Profile, on_delete=models.PROTECT, null=True,
related_name='center_creator', verbose_name='مـدیـر ایـجاد')

    manager = models.ForeignKey(to=Profile, on_delete=models.PROTECT,
related_name='center_manager',
                                verbose_name='مـدیـر مـرکـز')
    employees = models.ManyToManyField(to=Profile, blank=True, verbose_name=' اعضای
مـرکـز')

    def __str__(self):
        return f"{self.name}-{self.manager}"


def get_projectpack_attachment_directory_path(instance, filename):
    return 'projectpacks/%d_attachment_%s' % (int(time.time()), filename)


class ProjectPackAttachment(models.Model):
    file = models.FileField(upload_to=get_projectpack_attachment_directory_path,
verbose_name='فـایـل', max_length=۲۵۵)
    description = models.TextField(null=True, blank=True, verbose_name='تـوضیحات')
```

پک پروژه به عنوان مجموعه ای پروژه ها یک رابطه foreign key با مرکز دارد. علاوه بر آن توسط فیلد manager و monitoring_manager مدیر پک پروژه و کارشناس کنترل آن مشخص می شود. اطلاعات زمانی شروع و ددلاین و پایان پک و همچنین مبلغ کلی قرار داد و مبلغ تسویه شده نیز در این مدل نگهداری می شود.

```
class ProjectPack(models.Model):
    name = models.CharField(max_length=۲۵۵, verbose_name='نـام', unique=True)
    code = models.CharField(null=True, blank=True, max_length=۲۵۵, verbose_name='کـد')
    description = models.TextField(null=True, blank=True, verbose_name='تـوضیحات')

    created_at = models.DateTimeField(auto_now_add=True, verbose_name='تـاریـخ ایـجاد')

    manager = models.ForeignKey(to=Profile, on_delete=models.PROTECT,
related_name='projectpack_manager',
                                verbose_name='مـدیـر پـک پـروژه')
    monitoring_manager = models.ForeignKey(to=Profile, on_delete=models.PROTECT,
related_name='projectpack_monitoring_manager',
                                verbose_name='کـارشنـاس کـنـتـرل پـروژه')
    center = models.ForeignKey(to=Center, on_delete=models.PROTECT,
verbose_name='مـرکـز')
    started_at = models.DateTimeField(null=True, blank=True, verbose_name=' شروع
قـرارداد')
    to_be_finished = models.DateTimeField(verbose_name='ددلایـن قـرارداد')
    payment = models.IntegerField(verbose_name='مـبـلـغ قـرارداد')
    paid = models.IntegerField(default=۰, verbose_name='مـبـلـغ پـرداخت شده')
```

```
    usable_fund = models.IntegerField(default=۰, verbose_name='مبلغ قابل استفاده جهت تسویه پروژه')

    created_financial_statement = models.BooleanField(default=False, verbose_name='صورت مالی ایجاد شده؟')

    time_supplemented = models.BooleanField(default=False, verbose_name='مشمول متمم زمانی شده؟')
    time_supplement_days = models.IntegerField(default=۰, verbose_name='روزهای متمم شده')
    time_supplement_description = models.TextField(null=True, blank=True, verbose_name='توضیحات متمم زمانی')
    time_supplement_form_uploaded = models.BooleanField(default=False, verbose_name='فرم متمم زمانی بارگذاری شده؟')

    finished_at = models.DateTimeField(null=True, blank=True, verbose_name='تاریخ خاتمه قرارداد')

    status = models.CharField(max_length=۲۰, choices=STATUS_TYPES, default='to_do', verbose_name='وضعیت')

    attachments = models.ManyToManyField(ProjectPackAttachment, blank=True, verbose_name='فایل ها')

    def __str__(self):
        return f"{self.name}-{self.status}"

    def check_if_completed(self):
        for project in self.project_set.all():
            if project.status != 'completed':
                return False
        self.status = 'completed'
        self.save(update_fields=['status'])
        return True

    def check_if_verified(self):
        for project in self.project_set.all():
            if project.status != 'verified':
                return False
        self.status = 'verified'
        self.save(update_fields=['status'])
        return True


@receiver(post_save, sender=ProjectPack, dispatch_uid="projectpack_status_update")
def change_status(sender, instance, created, raw, update_fields, **kwargs):
    try:
        # create new log for new items
        if created:
            pass
        elif 'status' in update_fields:
            if instance.status == 'inprogress':
                instance.started_at = datetime.now()
                instance.finished_at = None
            elif instance.status == 'completed':
                instance.finished_at = None
            elif instance.status == 'to_do':
                instance.finished_at = None
            elif instance.status == 'verified':
                instance.finished_at = datetime.now()
            instance.save()
    except Exception:
        pass
```

```
def get_project_attachment_directory_path(instance, filename):
    return 'projects/%d_attachment_%s' % (int(time.time()), filename)


class ProjectAttachment(models.Model):
    file = models.FileField(upload_to=get_project_attachment_directory_path,
verbose_name='فایل', max_length=۲۵۵)
    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')
```

هر پروژه به عنوان مجموعه ای از تسک ها یک رابطه foreign key با مرکز دارد. علاوه بر آن برای اینکه مشخص شود پروژه زیر مجموعه کدام پک پروژه است یک رابطه foreign key با پک پروژه دارد. توسط فیلد manager و monitoring_manager پیمانکار و ناظر پروژه مشخص می شود. اطلاعات زمانی شروع و ددلاین و پایان پک و همچنین مبلغ کلی قرار داد و مبلغ تسویه شده، اطلاعات پیشرفت وزنی پروژه بر اساس تسک های زیر مجموعه نیز در این مدل نگهداری می شود.

```
class Project(models.Model):
    name = models.CharField(max_length=۲۵۵, verbose_name='نام', unique=True)
    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')

    created_at = models.DateTimeField(auto_now_add=True, verbose_name='تاریخ ایجاد')

    manager = models.ForeignKey(to=Profile, on_delete=models.PROTECT,
related_name='project_manager',
                                verbose_name='پیمانکار پروژه')
    monitoring_manager = models.ForeignKey(to=Profile, on_delete=models.PROTECT,
related_name='project_monitoring_manager',
                                verbose_name='ناظر پروژه')
    center = models.ForeignKey(to=Center, on_delete=models.PROTECT,
verbose_name='مرکز')
    project_pack = models.ForeignKey(to=ProjectPack, on_delete=models.PROTECT,
verbose_name='پک پروژه')
    employees = models.ManyToManyField(to=Profile, blank=True,
related_name='project_employees', verbose_name='اعضای پروژه')

    started_at = models.DateTimeField(null=True, blank=True, verbose_name=' شروع
قرارداد')
    to_be_finished = models.DateTimeField(verbose_name='ددلاین قرارداد')

    payment = models.IntegerField(verbose_name='مبلغ قرارداد')
    paid = models.IntegerField(default=۰, verbose_name='مبلغ پرداخت شده')
    usable_fund = models.IntegerField(default=۰, verbose_name=' مبلغ قابل استفاده جهت
تسویه تسک')

    created_financial_statement = models.BooleanField(default=False,
verbose_name='صورت مالی ایجاد شده؟')

    time_supplemented = models.BooleanField(default=False, verbose_name=' مشمول متمم
زمانی شده؟')
    time_supplement_days = models.IntegerField(default=۰, verbose_name=' روزهای متمم
شده')
    time_supplement_description = models.TextField(null=True, blank=True,
verbose_name='توضیحات متمم زمانی')
    time_supplement_form_uploaded = models.BooleanField(default=False,
verbose_name='فرم متمم زمانی بارگذاری شده؟')

    progress = models.IntegerField(default=۰, verbose_name='درصد پیشرفت وزنی')
```

```
    weightless_progress = models.IntegerField(default=۰, verbose_name='درصد پیشرفت')

    finished_at = models.DateTimeField(null=True, blank=True, verbose_name=' تاریخ
خاتمه قرارداد')

    status = models.CharField(max_length=۲۰, choices=STATUS_TYPES, default='to_do',
verbose_name='وضعیت')

    attachments = models.ManyToManyField(ProjectAttachment, blank=True,
verbose_name='فایل ها')

    def __str__(self):
        return f"{self.name}-{self.status}"

    def check_if_completed(self):
        for task in self.task_set.all():
            if task.status != 'completed':
                return False
        self.status = 'completed'
        self.save(update_fields=['status'])
        return True

    def check_if_verified(self):
        for task in self.task_set.all():
            if task.status != 'verified':
                return False
        self.status = 'verified'
        self.save(update_fields=['status'])
        return True

    def recalculate_weightless_progress(self):
        to_do = ۰
        for task in self.task_set.filter(status='to_do'):
            to_do += ۱
        inprogress = ۰
        for task in self.task_set.filter(status='inprogress'):
            inprogress += ۱
        completed = ۰
        for task in self.task_set.filter(status='completed'):
            completed += ۱
        verified = ۰
        for task in self.task_set.filter(status='verified'):
            verified += ۱

        total_task = to_do + inprogress + completed + verified

        if total_task == ۰:
            self.weightless_progress = ۰
        else:
            self.weightless_progress = round(۱۰۰ * (verified+completed) /
total_task,۲)
        self.save()

    def recalculate_progress(self):
        to_do = ۰
        for task in self.task_set.filter(status='to_do'):
            to_do += task.weight
        inprogress = ۰
        for task in self.task_set.filter(status='inprogress'):
            inprogress += task.weight
        completed = ۰
        for task in self.task_set.filter(status='completed'):
            completed += task.weight
```

```
        verified = ۰
        for task in self.task_set.filter(status='verified'):
            verified += task.weight

        total_task = to_do + inprogress + completed + verified

        if total_task == ۰:
            self.progress = ۰
        else:
            self.progress = round(۱۰۰ * (verified+completed) / total_task, ۲)
        self.save()


@receiver(post_save, sender=Project, dispatch_uid="project_status_update")
def change_status(sender, instance, created, raw, update_fields, **kwargs):
    try:
        # create new log for new items
        if created:
            pass
        elif 'status' in update_fields:
            if instance.status == 'inprogress':
                instance.started_at = datetime.now()
                instance.finished_at = None
            elif instance.status == 'completed':
                instance.finished_at = None
            elif instance.status == 'to_do':
                instance.finished_at = None
            elif instance.status == 'verified':
                instance.finished_at = datetime.now()
            instance.save()
    except Exception:
        pass


def get_task_attachment_directory_path(instance, filename):
    return 'tasks/%d_attachment_%s' % (int(time.time()), filename)


class TaskAttachment(models.Model):
    file = models.FileField(upload_to=get_task_attachment_directory_path,
verbose_name='فایل', max_length=۲۵۵)
    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')
```

هر تسکی که به کارمندی واگذار می شود، اطلاعات کارمند انجام دهنده آن توسط یک فیلد به پروفایل برمیگردد. علاوه بر اطلاعات زمانی تسک، وزن هر تسک و پروژه ای که تسک متعلق به آن است نیز در فیلدهای این مدل نگهداری می شوند. زیرتسک ها نیز از لحاظ ماهیت تفاوتی با تسک ها ندارند به جز اینکه فیلدی که مشخص کنند پروژه مالک است در اینجا تبدیل به فیلد تسک مادر می شود.

```
class Task(models.Model):
    name = models.CharField(max_length=۲۵۵, verbose_name='نام', unique=True)
    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')

    created_at = models.DateTimeField(auto_now_add=True, verbose_name='تاریخ ایجاد')

    employee = models.ForeignKey(null=True, blank=True, to=Profile,
on_delete=models.PROTECT,
                                 related_name='task_employee', verbose_name='کارمند')
    project = models.ForeignKey(null=True, blank=True, to=Project,
on_delete=models.PROTECT, verbose_name='پروژه')
```

```
    started_at = models.DateTimeField(null=True, blank=True, verbose_name=' شروع
قرارداد')
    to_be_finished = models.DateTimeField(null=True, blank=True, verbose_name=' ددلاین
قرارداد')

    payment = models.IntegerField(default=۰, verbose_name='مبلغ قرارداد')
    paid = models.IntegerField(default=۰, verbose_name='مبلغ پرداخت شده')

    created_financial_statement = models.BooleanField(default=False,
verbose_name='صورت مالی ایجاد شده؟')

    weight = models.IntegerField(default=۱, verbose_name='وزن')

    progress = models.IntegerField(default=۰, verbose_name='درصد پیشرفت')

    finished_at = models.DateTimeField(null=True, blank=True, verbose_name=' تاریخ
خاتمه قرارداد')

    status = models.CharField(max_length=۲۰, choices=STATUS_TYPES, default='to_do',
verbose_name='وضعیت')

    attachments = models.ManyToManyField(TaskAttachment, blank=True,
verbose_name='فایل ها')

    def __str__(self):
        return f"{self.name}-{self.project.name}-{self.status}"

    def check_if_completed(self):
        for task in self.subtask_set.all():
            if task.status != 'completed':
                return False
        self.status = 'completed'
        self.save(update_fields=['status'])
        return True

    def check_if_verified(self):
        for task in self.subtask_set.all():
            if task.status != 'verified':
                return False
        self.status = 'verified'
        self.save(update_fields=['status'])
        return True

    def recalculate_progress(self):
        to_do = ۰
        for subtask in self.subtask_set.filter(status='to_do'):
            to_do += subtask.weight
        inprogress = ۰
        for subtask in self.subtask_set.filter(status='inprogress'):
            inprogress += subtask.weight
        completed = ۰
        for subtask in self.subtask_set.filter(status='completed'):
            completed += subtask.weight
        verified = ۰
        for subtask in self.subtask_set.filter(status='verified'):
            verified += subtask.weight

        total_subtask = to_do + inprogress + completed + verified

        if total_subtask == ۰:
            self.progress = ۰
        else:
            self.progress = round(۱۰۰ * (verified+completed) / total_subtask, ۲)
```

```
        self.save()


@receiver(post_save, sender=Task, dispatch_uid="task_status_update")
def task_change_status(sender, instance, created, raw, update_fields, **kwargs):
    try:
        # create new log for new items
        if created:
            pass
        elif 'status' in update_fields:
            if instance.status == 'to_do':
                instance.finished_at = None
            elif instance.status == 'inprogress':
                instance.finished_at = None
                if not instance.started_at:
                    instance.started_at = datetime.now()
                instance.project.status = 'inprogress'
                instance.project.save(update_fields=['status'])
            elif instance.status == 'completed':
                instance.finished_at = None
                instance.project.check_if_completed()
            elif instance.status == 'verified':
                instance.finished_at = datetime.now()
                instance.project.check_if_verified()

            instance.save()
    except Exception:
        pass


def get_subtask_attachment_directory_path(instance, filename):
    return 'subtasks/%d_attachment_%s' % (int(time.time()), filename)


class SubTaskAttachment(models.Model):
    file = models.FileField(upload_to=get_subtask_attachment_directory_path,
verbose_name='فایل', max_length=۲۵۵)
    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')


class SubTask(models.Model):
    name = models.CharField(max_length=۲۵۵, verbose_name='نام', unique=True)
    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')

    created_at = models.DateTimeField(auto_now_add=True, verbose_name='تاریخ ایجاد')

    employee = models.ForeignKey(null=True, blank=True, to=Profile,
on_delete=models.PROTECT,
                                 related_name='subtask_employee', verbose_name=' مسئول
تسک')
    task = models.ForeignKey(to=Task, on_delete=models.PROTECT, verbose_name='تسک')

    started_at = models.DateTimeField(null=True, blank=True, verbose_name=' شروع
قرارداد')
    to_be_finished = models.DateTimeField(null=True, blank=True, verbose_name=' ددلاین
قرارداد')
    weight = models.IntegerField(default=۱, verbose_name='وزن')

    finished_at = models.DateTimeField(null=True, blank=True, verbose_name=' تاریخ
خاتمه قرارداد')

    status = models.CharField(max_length=۲۰, choices=STATUS_TYPES, default='to_do',
verbose_name='وضعیت')
```

```
    attachments = models.ManyToManyField(SubTaskAttachment,blank=True,
verbose_name='فایل ها')

    def __str__(self):
        return f"{self.name}-{self.task.name}-{self.task.project.name}-{self.status}"


@receiver(post_save, sender=SubTask, dispatch_uid="subtask_status_update")
def subtask_change_status(sender, instance, created, raw, update_fields, **kwargs):
    try:
        # create new log for new items
        if created:
            pass
        elif 'status' in update_fields:
            if instance.status == 'to_do':
                instance.finished_at = None
            elif instance.status == 'inprogress':
                instance.finished_at = None
                if not instance.started_at:
                    instance.started_at = datetime.now()
                instance.task.status = 'inprogress'
                instance.task.save(update_fields=['status'])
                instance.task.recalculate_progress()
            elif instance.status == 'completed':
                instance.finished_at = None
                instance.task.check_if_completed()
                instance.task.recalculate_progress()
            elif instance.status == 'verified':
                instance.finished_at = datetime.now()
                instance.task.check_if_verified()
                instance.task.recalculate_progress()

            instance.save()
    except Exception:
        pass
```

با توجه به کدها متوجه کلاس های attachment نیز می شوید که هر کدام از آن ها برای نگهداری فایل پیوست هر مدل شخصی سازی شده اند و محل ذخیره سازی هر کدام نیست توسط یک تابع تعیین می شود.

views

نماهای استفاده شده در این اپ از همه بیشتر است. هر مدل چهار عمل اصلی را در چهار نما دارد اما برای برخی از نماها به خاطر تفاوت در سطح دسترسی نقش های مختلف، دو نوع نمای مختلف نوشته شده است.

```
# centers
class CenterListView(LoginRequiredMixin, RoleMixin, PaginatedFilteredListView):
    model = Center
    template_name = 'centers/center_list.html'


class CenterCreateView(LoginRequiredMixin, RoleMixin, CreateView):
    model = Center
    template_name = 'centers/center_form.html'
    form_class = CenterForm
    success_url = reverse_lazy('centers:center_list')
    extra_context = {
        'tittle': 'افزودن',
```

```
    }

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)

        profiles = Profile.objects.all()
        context['profiles'] = []
        for profile in profiles:
            if profile.center_set.count() == ۰:
                context['profiles'].append(profile)

        centermanagers = Profile.objects.filter(permissions__code__in=['center'])
        context['center_managers'] = []
        for profile in centermanagers:
            try:
                Center.objects.get(manager=profile)
            except:
                context['center_managers'].append(profile)

        return context


class CenterDetailsView(LoginRequiredMixin, RoleMixin, DetailView):
    model = Center
    template_name = 'centers/center_detail.html'


class CenterUpdateForm(LoginRequiredMixin, RoleMixin, UpdateView):
    model = Center
    template_name = 'centers/center_form.html'
    form_class = CenterForm
    success_url = reverse_lazy('centers:center_list')
    extra_context = {
        'tittle': 'ویرایش',
    }

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)

        center = Center.objects.get(id=self.request.resolver_match.kwargs['pk'])

        profiles = Profile.objects.all()
        context['profiles'] = []
        for profile in profiles:
            if profile.center_set.count() == ۰:
                context['profiles'].append(profile)

        centermanagers = Profile.objects.filter(permissions__code__in=['center'])
        context['center_managers'] = []
        for profile in centermanagers:
            if profile.center_set.count() == ۰:
                context['center_managers'].append(profile)

        center = Center.objects.get(id=self.request.resolver_match.kwargs['pk'])
        for profile in center.employees.all():
            context['profiles'].append(profile)
        context['center_managers'].append(center.manager)

        return context


class CenterDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = Center
```

```
# project packs
class CenterProjectPackListView(LoginRequiredMixin, ListView):
    model = ProjectPack
    template_name = 'projectpacks/projectpack_list.html'

    def get_queryset(self):
        return
ProjectPack.objects.filter(center_id=self.request.resolver_match.kwargs['center_pk'])

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['the_center'] =
Center.objects.get(id=self.request.resolver_match.kwargs['center_pk'])
        context['in_center'] = True
        return context


class ProjectPackListView(LoginRequiredMixin, RoleMixin, PaginatedFilteredListView):
    model = ProjectPack
    template_name = 'projectpacks/projectpack_list.html'


class CenterProjectPackCreateView(LoginRequiredMixin, RoleMixin, CreateView):
    model = ProjectPack
    template_name = 'projectpacks/projectpack_form.html'
    form_class = ProjectPackForm
    extra_context = {
        'tittle': 'افزودن',
    }

    def get_success_url(self):
        # if you are passing 'pk' from 'urls' to 'DeleteView' for company
        # capture that 'pk' as companyid and pass it to 'reverse_lazy()' function
        return reverse_lazy('centers:center_projectpack_list', kwargs={'center_pk':
self.kwargs['center_pk'] })

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['centers'] =
Center.objects.filter(id=self.request.user.profile.get_user_center)
        context['the_center'] =
Center.objects.get(id=self.request.user.profile.get_user_center)
        context['in_center'] = True
        context['profiles'] = Profile.objects.all()
        context['projectpack_managers'] =
Profile.objects.filter(permissions__code__in=['projectpack'])
        context['projectpack_monitoring_managers'] =
Profile.objects.filter(permissions__code__in=['projectpack_monitoring'])
        return context


class ProjectPackCreateView(LoginRequiredMixin, RoleMixin, CreateView):
    model = ProjectPack
    template_name = 'projectpacks/projectpack_form.html'
    form_class = ProjectPackForm
    success_url = reverse_lazy('centers:projectpack_list')
    extra_context = {
        'tittle': 'افزودن',
    }

    def get_context_data(self, **kwargs):
```

```
        context = super().get_context_data(**kwargs)
        context['centers'] = Center.objects.all()
        context['in_center'] = False
        context['profiles'] = Profile.objects.all()
        context['projectpack_managers'] =
Profile.objects.filter(permissions__code__in=['projectpack'])
        context['projectpack_monitoring_managers'] =
Profile.objects.filter(permissions__code__in=['projectpack_monitoring'])
        return context


class ProjectPackAttachmentCreateView(LoginRequiredMixin, BSModalCreateView):
    template_name = 'projectpacks/attach_form.html'
    model = ProjectPackAttachment
    form_class = ProjectPackAttachmentForm

    def get_success_url(self):
        # if you are passing 'pk' from 'urls' to 'DeleteView' for company
        # capture that 'pk' as companyid and pass it to 'reverse_lazy()' function
        return reverse_lazy('centers:projectpack_details', kwargs={'pk':
self.kwargs['pk']})


class ProjectPackAttachmentDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = ProjectPackAttachment


class ProjectPackDetailsView(LoginRequiredMixin, RoleMixin, DetailView):
    model = ProjectPack
    template_name = 'projectpacks/projectpack_detail.html'


class CenterProjectPackUpdateView(LoginRequiredMixin, RoleMixin, UpdateView):
    model = ProjectPack
    template_name = 'projectpacks/projectpack_form.html'
    form_class = ProjectPackUpdateForm
    extra_context = {
        'tittle': 'افزودن',
    }

    def get_success_url(self):
        # if you are passing 'pk' from 'urls' to 'DeleteView' for company
        # capture that 'pk' as companyid and pass it to 'reverse_lazy()' function

        return reverse_lazy('centers:center_projectpack_list', kwargs={'center_pk':
self.kwargs['center_pk'] })

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['centers'] =
Center.objects.filter(id=self.request.user.profile.get_user_center)
        context['the_center'] =
Center.objects.get(id=self.request.user.profile.get_user_center)
        context['in_center'] = True
        context['profiles'] = Profile.objects.all()
        context['projectpack_managers'] =
Profile.objects.filter(permissions__code__in=['projectpack'])
        context['projectpack_monitoring_managers'] =
Profile.objects.filter(permissions__code__in=['projectpack_monitoring'])
        return context


class ProjectPackUpdateView(LoginRequiredMixin, RoleMixin, UpdateView):
```

```
    model = ProjectPack
    template_name = 'projectpacks/projectpack_form.html'
    form_class = ProjectPackUpdateForm
    success_url = reverse_lazy('centers:projectpack_list')
    extra_context = {
        'tittle': 'ویرایش',
    }

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['centers'] = Center.objects.all()
        context['in_center'] = False
        context['profiles'] = Profile.objects.all()
        context['projectpack_managers'] =
Profile.objects.filter(permissions__code__in=['projectpack'])
        context['projectpack_monitoring_managers'] =
Profile.objects.filter(permissions__code__in=['projectpack_monitoring'])
        return context


class ProjectPackDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = ProjectPack


# projects
class ProjectPackProjectListView(LoginRequiredMixin, RoleMixin,
PaginatedFilteredListView):
    model = Project
    template_name = 'projects/project_list.html'

    def get_queryset(self):
        return
Project.objects.filter(project_pack_id=self.request.resolver_match.kwargs['projectpack
_pk'])

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['the_projectpack'] =
ProjectPack.objects.get(id=self.request.resolver_match.kwargs['projectpack_pk'])
        return context


class ProjectPackProjectCreateView(LoginRequiredMixin, RoleMixin, CreateView):
    model = Project
    template_name = 'projects/project_form.html'
    form_class = ProjectForm
    extra_context = {
        'tittle': 'افزودن',
    }

    def get_success_url(self):
        return reverse_lazy('centers:projectpack_project_list',
kwargs={'projectpack_pk': self.object.project_pack.id })


    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        projectpack =
ProjectPack.objects.get(id=self.request.resolver_match.kwargs['projectpack_pk'])
        context['projectpacks'] =
ProjectPack.objects.filter(id=self.request.resolver_match.kwargs['projectpack_pk'])
        context['the_projectpack'] = projectpack
        context['in_projectpack'] = True
```

```
        context['centers'] = [projectpack.center]
        context['center'] = projectpack.center
        context['profiles'] = Profile.objects.all()
        context['project_managers'] =
Profile.objects.filter(center__in=[projectpack.center],permissions__code__in=['project
'])
        context['project_monitoring_managers'] =
Profile.objects.filter(center__in=[projectpack.center],permissions__code__in=['project
_monitoring'])
        return context


class ProjectCreateView(LoginRequiredMixin, RoleMixin, CreateView):
    model = Project
    template_name = 'projects/project_form.html'
    form_class = ProjectForm
    success_url = reverse_lazy('centers:project_list')
    extra_context = {
        'tittle': 'افزودن',
    }

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['centers'] =
Center.objects.filter(id=self.request.user.profile.get_user_center)
        context['in_center'] = False
        context['profiles'] = Profile.objects.all()
        context['project_managers'] = Profile.objects.filter(
            Q(is_projectmanager=True) | Q(is_centermanager=True) |
Q(is_supermanager=True)).distinct()
        return context


class ProjectBulkCreateView(LoginRequiredMixin, RoleMixin, FormView):
    template_name = 'projects/project_import.html'
    form_class = ProjectImportForm
    success_url = reverse_lazy('centers:project_list')
    extra_context = {
        'tittle': 'project',
    }


class TaskBulkCreateView(LoginRequiredMixin, RoleMixin, FormView):
    template_name = 'projects/project_import.html'
    form_class = TaskImportForm
    success_url = reverse_lazy('centers:project_list')
    extra_context = {
        'tittle': 'task',
    }


class ProjectAttachmentCreateView(LoginRequiredMixin, BSModalCreateView):
    template_name = 'projects/attach_form.html'
    model = ProjectAttachment
    form_class = ProjectAttachmentForm

    def get_success_url(self):
        # if you are passing 'pk' from 'urls' to 'DeleteView' for company
        # capture that 'pk' as companyid and pass it to 'reverse_lazy()' function
        return reverse_lazy('centers:projectpack_project_details', kwargs={'pk':
self.kwargs['pk'],'projectpack_pk': self.kwargs['projectpack_pk']})
```

```
class ProjectAttachmentDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = ProjectAttachment


class ProjectPackProjectDetailsView(LoginRequiredMixin, RoleMixin, DetailView):
    model = Project
    template_name = 'projects/project_detail.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)

        project = Project.objects.get(id=self.kwargs['pk'])

        if project.started_at:
            context['gone_days'] = (datetime.datetime.now().timestamp() -
project.started_at.timestamp()) / ٨٦٤٠٠
        else:
            context['gone_days'] = ٠
        context['remained_days'] = (project.to_be_finished.timestamp() -
datetime.datetime.now().timestamp()) / ٨٦٤٠٠

        project.recalculate_progress()
        project.recalculate_weightless_progress()

        if project.started_at:
            total_duration = (project.to_be_finished.timestamp() -
project.started_at.timestamp()) / ٨٦٤٠٠
            context['done_days'] = project.progress*total_duration/١٠٠
            context['programmed_remained_days'] = total_duration*(١٠٠-
project.progress)/١٠٠
        else:
            context['done_days'] = ٠
            context['programmed_remained_days'] = context['remained_days']


        return context


class ProjectPackProjectUpdateForm(LoginRequiredMixin, RoleMixin, UpdateView):
    model = Project
    template_name = 'projects/project_form.html'
    form_class = ProjectUpdateForm
    extra_context = {
        'tittle': 'ویرایش',
    }

    def get_success_url(self):
        return reverse_lazy('centers:projectpack_project_list',
kwargs={'projectpack_pk': self.object.project_pack.id })

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        projectpack =
ProjectPack.objects.get(id=self.request.resolver_match.kwargs['projectpack_pk'])
        context['projectpacks'] =
ProjectPack.objects.filter(id=self.request.resolver_match.kwargs['projectpack_pk'])
        context['the_projectpack'] = projectpack
        context['in_projectpack'] = True
        context['centers'] = [projectpack.center]
        context['center'] = projectpack.center
        context['profiles'] = Profile.objects.all()
        context['project_managers'] =
Profile.objects.filter(center__in=[projectpack.center],
```

```
permissions__code__in=['project'])
        context['project_monitoring_managers'] =
Profile.objects.filter(center__in=[projectpack.center],

permissions__code__in=['project_monitoring'])
        return context


class ProjectPackProjectDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = Project


class TaskUpdateView(LoginRequiredMixin, BSModalUpdateView):
    template_name = 'tasks/task_form.html'
    model = Task
    form_class = TaskForm

    def get_success_url(self):
        # if you are passing 'pk' from 'urls' to 'DeleteView' for company
        # capture that 'pk' as companyid and pass it to 'reverse_lazy()' function
        project = self.object.project
        return reverse_lazy('centers:task_list', kwargs={'pk_project': project.id})


class TaskAttachmentCreateView(LoginRequiredMixin, BSModalCreateView):
    template_name = 'tasks/attach_form.html'
    model = TaskAttachment
    form_class = TaskAttachmentForm

    def get_success_url(self):
        # if you are passing 'pk' from 'urls' to 'DeleteView' for company
        # capture that 'pk' as companyid and pass it to 'reverse_lazy()' function
        project = Task.objects.get(id=self.kwargs['pk']).project
        return reverse_lazy('centers:task_list', kwargs={'pk_project': project.id})


class SubTaskUpdateView(LoginRequiredMixin, BSModalUpdateView):
    template_name = 'tasks/subtask_form.html'
    model = SubTask
    form_class = SubTaskForm

    def get_success_url(self):
        # if you are passing 'pk' from 'urls' to 'DeleteView' for company
        # capture that 'pk' as companyid and pass it to 'reverse_lazy()' function
        project = self.object.task.project
        return reverse_lazy('centers:task_list', kwargs={'pk_project': project.id})


class SubTaskAttachmentCreateView(LoginRequiredMixin, BSModalCreateView):
    template_name = 'tasks/attach_form.html'
    model = SubTaskAttachment
    form_class = SubTaskAttachmentForm

    def get_success_url(self):
        # if you are passing 'pk' from 'urls' to 'DeleteView' for company
        # capture that 'pk' as companyid and pass it to 'reverse_lazy()' function
        project = SubTask.objects.get(id=self.kwargs['pk']).task.project
        return reverse_lazy('centers:task_list', kwargs={'pk_project': project.id})


# tasks
class TaskListView(LoginRequiredMixin, PaginatedFilteredListView):
```

```
    model = Task
    template_name = 'tasks/task_list.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        project = Project.objects.get(id=self.kwargs['pk_project'])
        subtasks = SubTask.objects.filter(task__project_id=project.id)

        context['now'] = datetime.datetime.now()

        context['employees'] = project.employees.all()
        context['project_id'] = project.id
        context['project'] = project

        context['task_to_do'] = project.task_set.filter(status='to_do')
        context['subtask_to_do'] = subtasks.filter(status='to_do')

        context['task_inprogress'] =
project.task_set.filter(status__exact='inprogress')
        context['subtask_inprogress'] = subtasks.filter(status__exact='inprogress')

        context['task_completed'] = project.task_set.filter(status__exact='completed')
        context['subtask_completed'] = subtasks.filter(status__exact='completed')

        context['task_verified'] = project.task_set.filter(status__exact='verified')
        context['subtask_verified'] = subtasks.filter(status__exact='verified')

        return context


class TaskDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = Task


class SubTaskDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = SubTask
```

در نهایت یک نمای ajax نوشته شده است تا تغییرات در لحظه ای مورد نیاز در هر صفحه تمپلیت را انجام دهد. هر درخواست ajax از صفحه فرانت اند با یک نوع request_type به بکند می آید که پس از شکسته شدن نوعش در هر شرط چک شده و عملیاتش انجام می شود. عموم دستورات ایجکسی برای تغییرات یک یا دو فیلد از کل مدل مورد استفاده قرار میگیرند. البته برای ساخت تسک از نمای دیفالت ساخت تسک استفاده نشده و با استفاده از درخواست task_create و با داده های POST شده به همراه درخواست، تسک ساخته می شود.

```
class AjaxHandler(TemplateView):
    def get(self, request, *args, **kwargs):
        request_type = request.GET.get('request_type')
        data = {'error': ·}

        if request_type == 'projectpack_suppliment_time':
            projectpack_id = request.GET.get('projectpack_id')
            time_supplement_days = int(request.GET.get('time_supplement_days'))
            time_supplement_description =
request.GET.get('time_supplement_description')

            projectpack = ProjectPack.objects.get(id=projectpack_id)
            projectpack.time_supplemented = True
            projectpack.time_supplement_days = time_supplement_days
            projectpack.time_supplement_description = time_supplement_description
```

```
            new_date = projectpack.to_be_finished
            new_date += datetime.timedelta(days=time_supplement_days)

            projectpack.to_be_finished = new_date
            projectpack.save()

        if request_type == 'project_suppliment_time':
            project_id = request.GET.get('project_id')
            time_supplement_days = int(request.GET.get('time_supplement_days'))
            time_supplement_description =
request.GET.get('time_supplement_description')

            project = Project.objects.get(id=project_id)
            project.time_supplemented = True
            project.time_supplement_days = time_supplement_days
            project.time_supplement_description = time_supplement_description

            new_date = project.to_be_finished
            new_date += datetime.timedelta(days=time_supplement_days)

            project.to_be_finished = new_date
            project.save()

        if request_type == 'profile_password_change':
            profile_id = request.GET.get('profile_id')
            new_password = request.GET.get('new_password')
            profile = Profile.objects.get(id=profile_id)
            profile.user.set_password(new_password)
            profile.user.save()

        if request_type == 'task_payment':
            task_id = request.GET.get('task_id')
            payment = request.GET.get('payment')
            task = Task.objects.get(id=task_id)
            if task.paid + int(payment) > task.payment:
                data['error'] = ١
                return JsonResponse(data)
            task.paid += int(payment)
            task.save()
            if task.project.paid + int(payment) > task.project.payment:
                data['error'] = ٢
                return JsonResponse(data)
            task.project.paid += int(payment)
            task.project.save()

        if request_type == 'task_edit':
            task_id = request.GET.get('task_id')
            payment = int(request.GET.get('payment'))
            task = Task.objects.get(id=task_id)
            project = task.project
            sum = ٠
            for tsk in project.task_set.all():
                if tsk != task:
                    sum += task.payment
            if (project.payment-sum)>=payment:
                task.payment = payment
                task.save()
            else:
                data['error'] = ٣
                return JsonResponse(data)

        if request_type == 'task_create':
```

```
            project_id = request.GET.get('project_id')
            task_name = request.GET.get('task_name')
            if task_name:
                Task.objects.create(name=task_name,
project=Project.objects.get(id=project_id))

        if request_type == 'task_status_update':
            task_id = request.GET.get('task_id')
            task_new_status = request.GET.get('task_new_status')
            task = Task.objects.get(id=task_id)
            task.status = task_new_status
            task.save(update_fields=['status'])

        if request_type == 'subtask_create':
            task_id = request.GET.get('task_id')
            subtask_name = request.GET.get('subtask_name')

            task = Task.objects.get(id=task_id)
            if subtask_name:
                subtask = SubTask.objects.create(name=subtask_name, task=task)
                subtask.employee = task.employee
                subtask.save()

        if request_type == 'subtask_status_update':
            subtask_id = request.GET.get('subtask_id')
            subtask_new_status = request.GET.get('subtask_new_status')
            subtask = SubTask.objects.get(id=subtask_id)
            subtask.status = subtask_new_status
            subtask.save(update_fields=['status'])

        return JsonResponse(data)
```

equipments

models

تجهیزات در سه مدل کلی نوع تجهیزات، خود تجهیزات و فرآیند اجاره کردن خلاصه شده است. هرکدام از این مدل ها نیز اطلاعات مورد نیاز را به صورت رکوردهایی در دیتابیس ذخیره سازی می کنند.

```
EQUIPMENT_TYPES = [
    ('car', 'ماشین'),
    ('pc', 'رایانه'),
    ('laptop', 'رایانه همراه'),
    ('tablet', 'تبلت'),
    ('mobile', 'تلفن همراه'),
]


class EquipmentType(models.Model):
    name = models.CharField(max_length=۲۵۵, verbose_name='نام', unique=True)
    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')

    created_at = models.DateTimeField(auto_now_add=True, verbose_name='تاریخ ایجاد')

    def __str__(self):
        return f"{self.name}-{self.description}"


class Equipment(models.Model):
    name = models.CharField(max_length=۲۵۵, verbose_name='نام', unique=True)
```

```
    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')

    created_at = models.DateTimeField(auto_now_add=True, verbose_name='تاریخ ایجاد')

    owner = models.ForeignKey(to=Center, on_delete=models.PROTECT,
verbose_name='مرکز')

    type = models.ForeignKey(to=EquipmentType, on_delete=models.PROTECT,
verbose_name='نوع')

    is_rented = models.BooleanField(default=False, verbose_name='اجاره؟')

    def __str__(self):
        return f"{self.name}-{self.owner.name}"

    def check_if_out_of_rent(self):
        now = datetime.datetime.now()
        rents = Rent.objects.filter(equipment=self, at__gte=now, to__lte=now)
        if rents.count():
            self.is_rented = True
            for rent in rents:
                rent.is_reservation = False
            return False
        self.is_rented = False
        self.save()
        return True
```

اجاره کردن یکی از تجهیزات به این صورت است که در یک فیلد کاربر اجاره کننده، در یک فیلد خود دستگاه و در فیلد دیگر مرکز مالک دستگاه ذخیره می شود. علاوه بر نگهداری اطلاعات زمانی، اطلاعات مالی نیز در مدل rent ذخیره سازی می شوند.

```
class Rent(models.Model):
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='تاریخ ایجاد')

    equipment = models.ForeignKey(to=Equipment, on_delete=models.PROTECT,
verbose_name='تجهیزات')

    borrower = models.ForeignKey(to=Profile, on_delete=models.PROTECT,
verbose_name='اجاره کننده')
    center = models.ForeignKey(to=Center, on_delete=models.PROTECT,
verbose_name='مرکز')
    project = models.ForeignKey(to=Project, on_delete=models.PROTECT,
verbose_name='پروژه')

    at = models.DateTimeField(verbose_name='شروع قرارداد اجاره')
    to = models.DateTimeField(verbose_name='ددلاین قرارداد اجاره')

    payment = models.FloatField(default=۰, verbose_name='مبلغ قرارداد اجاره')
    paid = models.FloatField(default=۰, verbose_name='مبلغ پرداخت شده')

    returned_at = models.DateTimeField(null=True, blank=True, verbose_name='تاریخ
خاتمه قرارداد')

    is_reservation = models.BooleanField(default=True, verbose_name='فرآیند رزرو؟')

    def __str__(self):
        return f"{self.borrower}-{self.center.name}-{self.project.name}"
```

views

همانند اپ قبلی هر مدل چهار عمل اصلی را در چهار نما دارد اما برای برخی از نماها به خاطر تفاوت در سطح دسترسی نقش های مختلف، دو نوع نمای مختلف نوشته شده است.

```
# equipments
class CenterEquipmentListView(LoginRequiredMixin, ListView):
    model = Equipment
    template_name = 'equipment_list.html'
    extra_context = {
        'in_center': True,
    }

    def get_queryset(self):
        return
Equipment.objects.filter(owner_id=self.request.resolver_match.kwargs['center_pk'])


class EquipmentListView(LoginRequiredMixin, ListView):
    model = Equipment
    template_name = 'equipment_list.html'


class CenterEquipmentCreateView(LoginRequiredMixin, RoleMixin, CreateView):
    model = Equipment
    template_name = 'equipment_form.html'
    form_class = EquipmentForm
    success_url = reverse_lazy('equipments:equipment_list')
    extra_context = {
        'tittle': 'افزودن',
    }

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['centers'] =
Center.objects.filter(id=self.request.resolver_match.kwargs['center_pk'])
        context['in_center'] = True
        context['types'] = EquipmentType.objects.all()

        return context


class EquipmentCreateView(LoginRequiredMixin, RoleMixin, CreateView):
    model = Equipment
    template_name = 'equipment_form.html'
    form_class = EquipmentForm
    success_url = reverse_lazy('equipments:equipment_list')
    extra_context = {
        'tittle': 'افزودن',
    }

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)

        context['centers'] = Center.objects.all()
        context['types'] = EquipmentType.objects.all()

        return context


class EquipmentDetailsView(LoginRequiredMixin, RoleMixin, DetailView):
    model = Equipment
```

```
    template_name = 'equipment_detail.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)

        self.object.check_if_out_of_rent()

        context['events'] =
Rent.objects.filter(equipment_id=self.request.resolver_match.kwargs['pk'])
        context['projects'] = Project.objects.all()
        context['employees'] = Profile.objects.all()
        context['centers'] = Center.objects.all()

        profile = self.request.user.profile
        context['center'] = profile.center_set.first()

        return context


class EquipmentUpdateForm(LoginRequiredMixin, RoleMixin, UpdateView):
    model = Equipment
    template_name = 'equipment_form.html'
    form_class = EquipmentForm
    success_url = reverse_lazy('equipments:equipment_list')
    extra_context = {
        'tittle': 'ویرایش',
    }

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)

        context['centers'] = Center.objects.all()

        context['types'] = EquipmentType.objects.all()

        return context


class EquipmentDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = Equipment
```

در نهایت یک نمای ajax نوشته شده است تا تغییرات در لحظه ای مورد نیاز در هر صفحه تمپلیت را انجام دهد. هر درخواست ajax از صفحه فرانت اند با یک نوع request_type به بکند می آید که پس از شکسته شدن نوعش در هر شرط چک شده و عملیاتش انجام می شود. انواع درخواست ها عبارتند از ایجاد و ویرایش نوع تجهیزات و فرآیند رزرو کردن تجهیزات.

```
class AjaxHandler(TemplateView):
    def get(self, request, *args, **kwargs):
        request_type = request.GET.get('request_type')
        data = {'error': ۰}

        if request_type =='create_equipmenttype':
            name = request.GET.get('name')
            description = request.GET.get('description')
            EquipmentType.objects.get_or_create(name=name, description=description)

        if request_type =='delete_equipmenttype':
            equipmenttype_id = int(request.GET.get('equipmenttype_id'))
            equipmenttype = EquipmentType.objects.get(id=equipmenttype_id)
            equipmenttype.delete()
```

```
        if request_type =='rent':
            equipment_id = request.GET.get('equipment_id')
            project_id = request.GET.get('project_id')
            center_id = request.GET.get('center_id')
            borrower_id = request.GET.get('borrower_id')
            at = request.GET.get('at')
            to = request.GET.get('to')

            if at:
                at = datetime.datetime.strptime(at, "%Y-%m-%d %H:%M")
            if to:
                to = datetime.datetime.strptime(to, "%Y-%m-%d %H:%M")

            if at >= to:
                data['error'] = ۳
                return JsonResponse(data)
            now = datetime.datetime.now()
            if now > at or now > to:
                data['error'] = ۲
                return JsonResponse(data)

            equipment = Equipment.objects.get(id=equipment_id)
            rents = Rent.objects.filter(equipment=equipment).filter(Q(to__range=[at,
to]) or Q(at__range=[at, to]) or Q(Q(at__gte=at,to__lte=at) and
Q(at__gte=to,to__lte=to)))
            if rents.count():
                data['error'] = ۱
                return JsonResponse(data)

            borrower = Profile.objects.get(id=borrower_id)
            center = Center.objects.get(id=center_id)
            project = Project.objects.get(id=project_id)

            if equipment.is_rented:
                Rent.objects.create(equipment=equipment, project=project,
borrower=borrower, center=center, at=at, to=to)
            else:
                Rent.objects.create(equipment=equipment, project=project,
borrower=borrower, center=center, at=at, to=to)

        return JsonResponse(data)
```

transactions

models

در مدل های تراکنش های مالی به ازای هر فرآیند، یک پروژه و پروژه یک تراکنش مالی ایجاد می شود که تمام رکوردهای مورد نظر و متناظر با آن را ذخیره می کند. هر تراکنش مالی نیز امکان پیوست فایل دارد و علاوه بر نگهداری اطلاعات مالی و زمانی، در یک فیلد مرکزی که مالک تراکنش نیز هست را نگه داری میکند. اما مدل های اصلی قابل استفاده مدل هایی هستند که متناسب با هر مدل یک پروژه، پروژه و تسک از مدل اصلی تراکنش به ارث برده شده اند که به صورت یک نشانه گر خارجی شی مورد نظر را که تراکنش به صورت ویژه برای آن ایجاد شده نگهداری می کنند.

```
STATUS_TYPES = [
    ('to_do', 'جهت انجام'),
    ('inprogress', 'در حال انجام'),
    ('completed', 'انجام شده'),
    ('verified', 'تایید شده'),
]

TRANSACTION_TITTLES = [
    ('prepayment', 'پیش پرداخت'),
    ('instalment', 'قسط'),
    ('paragraph', 'بند'),
    ('final_instalment', 'حسن انجام کار'),
]


def get_transaction_attachment_directory_path(instance, filename):
    return 'transactions/%d_attachment_%s' % (int(time.time()), filename)


class TransactionAttachment(models.Model):
    file = models.FileField(upload_to=get_transaction_attachment_directory_path,
verbose_name='فایل', max_length=۲۵۵)
    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')


class Transaction(PolymorphicModel):
    tittle = models.CharField(max_length=۲۰, choices=TRANSACTION_TITTLES,
verbose_name='نوع')

    center = models.ForeignKey(to=Center, on_delete=models.PROTECT,
verbose_name='مرکز')

    value = models.IntegerField(default=۰, verbose_name='مبلغ تراکنش')

    sequence_number = models.IntegerField(default=۰, verbose_name=' شماره توالی
تراکنش')

    due_flag = models.BooleanField(default=False, verbose_name=' قابل پرداخت بودن
تراکنش')

    due_progress = models.IntegerField(default=۰, verbose_name='موعد وزنی پرداخت')

    paid_at = models.DateTimeField(null=True, blank=True, verbose_name=' تاریخ پرداخت
شده')

    description = models.TextField(null=True, blank=True, verbose_name='توضیحات')

    attachments = models.ManyToManyField(TransactionAttachment, blank=True,
verbose_name='فایل ها')


class ProjectPackTransaction(Transaction):
    project_pack = models.ForeignKey(to=ProjectPack, on_delete=models.PROTECT,
verbose_name='پک پروژه')


class ProjectTransaction(Transaction):
    project = models.ForeignKey(to=Project, on_delete=models.PROTECT,
verbose_name='پروژه')


class TaskTransaction(Transaction):
    task = models.ForeignKey(to=Task, on_delete=models.PROTECT, verbose_name='تسک')
```

views

نماهای تراکنش ها نیز علاوه بر چهار نوع عمل اصلی ایجاد، اصلاح، حذف و لیست کردن یک کلاس کنترل کننده ajax دارد که بسیاری از عملیات های تغییرات تراکنش ها نیز توسط این نما کنترل می شود. در نمای کنترل کننده ایجکسی ترتیب توالی پرداخت اقساط و ایجاد صورت مالی و لیست تراکنش برای هر کدام از پک پروژه ها و پروژه ها و تسک ها توسط درخواست ها با انواع request_type انجام می شوند.

```
class CenterTransactionListView(LoginRequiredMixin, ListView):
    model = Transaction
    template_name = 'transaction_list.html'

    def get_queryset(self):
        return
Transaction.objects.filter(center_id=self.request.resolver_match.kwargs['center_pk'])

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['projectpack_transactions'] =
ProjectPackTransaction.objects.filter(center_id=self.request.resolver_match.kwargs['ce
nter_pk'])
        context['project_transactions'] =
ProjectTransaction.objects.filter(center_id=self.request.resolver_match.kwargs['center
_pk'])
        context['the_center'] =
Center.objects.get(id=self.request.resolver_match.kwargs['center_pk'])
        context['in_center'] = True
        return context


class TransactionListView(LoginRequiredMixin, ListView):
    model = Transaction
    template_name = 'transaction_list.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['projectpack_transactions'] = ProjectPackTransaction.objects.all()
        context['project_transactions'] = ProjectTransaction.objects.all()
        return context


class TransactionDeleteView(LoginRequiredMixin, RoleMixin, JSONDeleteView):
    model = Transaction


class TransactionAttachmentCreateView(LoginRequiredMixin, BSModalCreateView):
    template_name = 'attach_form.html'
    model = TransactionAttachment
    form_class = TransactionAttachmentForm

    def get_success_url(self):
        # if you are passing 'pk' from 'urls' to 'DeleteView' for company
        # capture that 'pk' as companyid and pass it to 'reverse_lazy()' function
        transaction = Transaction.objects.get(id=self.kwargs['pk'])
        return reverse_lazy('transactions:center_transaction_list',
kwargs={'center_pk': transaction.center.id})
```

```
class AjaxHandler(TemplateView):
    def is_sorted(self, tmp):
        flag = True
        i = ١
        while i < len(tmp):
            if tmp[i] < tmp[i - ١]:
                flag = False
                return flag
            i += ١
        return flag

    def get(self, request, *args, **kwargs):
        request_type = request.GET.get('request_type')
        data = {'error': ٠}

        if request_type == 'projectpack_create_financial_statement':
            projectpack_id = int(request.GET.get('projectpack_id'))
            projectpack = ProjectPack.objects.get(id=projectpack_id)
            prepayment = int(request.GET.get('prepayment'))

            installment_dues = request.GET.get('installment_dues').split(';')[:-١]
            installment_num = int(request.GET.get('installment_num'))

            if prepayment == ٠:
                data = {'error': ١}
                return JsonResponse(data)

            if installment_num == ٠:
                data = {'error': ٣}
                return JsonResponse(data)

            if not self.is_sorted(installment_dues):
                data = {'error': ٤}
                return JsonResponse(data)

            if projectpack.payment == ٠:
                data = {'error': ٢}
                return JsonResponse(data)

            ProjectPackTransaction.objects.create(project_pack=projectpack,
center=projectpack.center, tittle=TRANSACTION_TITTLES[٠][٠], value=prepayment,
due_progress=٠, sequence_number=٠, due_flag=True)

            final_instalment = projectpack.payment*٠,١
            instalment_value = (projectpack.payment*٠,٩ - prepayment)/installment_num

            for i in range(installment_num):
                ProjectPackTransaction.objects.create(project_pack=projectpack,
center=projectpack.center, tittle=TRANSACTION_TITTLES[١][٠],
value=instalment_value,sequence_number=i+١,due_progress=installment_dues[i])

            ProjectPackTransaction.objects.create(project_pack=projectpack,
center=projectpack.center,
                                                  tittle=TRANSACTION_TITTLES[٣][٠],
value=final_instalment, due_progress=١٠٠,
                                                  sequence_number=installment_num+١)

            projectpack.created_financial_statement = True
            projectpack.save()

        if request_type == 'project_create_financial_statement':
```

```
            project_id = int(request.GET.get('project_id'))
            prepayment = int(request.GET.get('prepayment'))
            project = Project.objects.get(id=project_id)

            paragraph_dues = request.GET.get('paragraph_dues').split(';')[:-١]
            paragraph_num = int(request.GET.get('paragraph_num'))
            if paragraph_num == ٠:
                data = {'error': ٣}
                return JsonResponse(data)

            if not self.is_sorted(paragraph_dues):
                data = {'error': ٤}
                return JsonResponse(data)

            if project.payment == ٠:
                data = {'error': ٢}
                return JsonResponse(data)

            flag = True
            if prepayment > ٠:
                ProjectTransaction.objects.create(project=project,
center=project.center, tittle=TRANSACTION_TITTLES[٠][٠], value=prepayment,
due_progress=٠,sequence_number=٠, due_flag=True)
                flag = False

            paragraph_value = (project.payment - prepayment)/paragraph_num
            for i in range(paragraph_num):
                if i == ٠ and flag:
                    ProjectTransaction.objects.create(project=project,
center=project.center, tittle=TRANSACTION_TITTLES[٢][٠],
value=paragraph_value,sequence_number=i+١,due_progress=paragraph_dues[i],due_flag=True
)
                else:
                    ProjectTransaction.objects.create(project=project,
center=project.center, tittle=TRANSACTION_TITTLES[٢][٠],
value=paragraph_value,sequence_number=i+١,due_progress=paragraph_dues[i])

            project.created_financial_statement = True
            project.save()

        if request_type == 'task_create_financial_statement':
            task_id = int(request.GET.get('task_id'))
            task = Task.objects.get(id=task_id)

            paragraph_num = int(request.GET.get('paragraph_num'))
            if paragraph_num == ٠:
                data = {'error': ٣}
                return JsonResponse(data)

            if task.payment == ٠:
                data = {'error': ٢}
                return JsonResponse(data)

            now = datetime.datetime.now()
            paragraph_value = task.payment/paragraph_num
            for i in range(paragraph_num):
                try:
                    onemonthafterthat = datetime.datetime(year=now.year,
month=now.month+i+١, day=now.day)
                except:
                    onemonthafterthat = datetime.datetime(year=now.year + ١,
month=i+١, day=now.day)
```

```
                TaskTransaction.objects.create(task=task, center=task.project.center,
tittle=TRANSACTION_TITTLES[۲][۰],
value=paragraph_value,sequence_number=i+۱,due_date=onemonthafterthat)

            task.created_financial_statement = True
            task.save()

        if request_type == 'pay_financial_statement':
            transaction_id = int(request.GET.get('transaction_id'))
            transaction = Transaction.objects.get(id=transaction_id)
            now = datetime.datetime.now()

            try:
                projectpack = transaction.project_pack
                projectpack.usable_fund = transaction.value
                projectpack.paid += transaction.value
                if transaction.sequence_number == ۰:
                    projectpack.started_at = now
                projectpack.save()

                # transaction.due_flag = False
                # transaction.save()
                try:
                    next_projectpack_transaction =
projectpack.projectpacktransaction_set.get(sequence_number=transaction.sequence_number
+ ۱)
                    next_projectpack_transaction.due_flag = True
                    next_projectpack_transaction.save()
                except:
                    pass
            except:
                try:
                    project = transaction.project
                    value = transaction.value
                    usable_fund = project.project_pack.usable_fund
                    if value <= usable_fund:
                        project.project_pack.usable_fund -= value
                        project.project_pack.save()
                        project.usable_fund += value
                        project.paid += value
                        project.save()

                        # transaction.due_flag = False
                        # transaction.save()

                        try:
                            next_project_transaction =
project.projecttransaction_set.get(sequence_number=transaction.sequence_number + ۱)
                            next_project_transaction.due_flag = True
                            next_project_transaction.save()
                        except:
                            pass


                    else:
                        data = {'error': ۱}
                        return JsonResponse(data)
                except:
                    task = transaction.task
                    value = transaction.value
                    usable_fund = task.project.usable_fund
                    if value <= usable_fund:
                        task.project.usable_fund -= value
```

```
                task.project.save()
                task.paid += transaction.value
                task.save()
            else:
                data = {'error': ٢}
                return JsonResponse(data)


    transaction.paid_at = now
    transaction.due_flag = False
    transaction.save()

return JsonResponse(data)
```