**Company Hierarchy Project**

**Project Setup (Hierarchy)**

**1.1. Create virtual environment and install Django & DRF**

python -m venv env

env\Scripts\activate # Use source env/bin/activate on Linux

pip install django djangorestframework

**1.2. Create a Django project**

django-admin startproject CompanyProject

cd CompanyProject

**1.3. Create the app**

python manage.py startapp staff

**1.4. Add to settings.py**

# CompanyProject/settings.py

INSTALLED\_APPS = [

...

'rest\_framework',

'staff',

]

**Step 2: Create Models**

**staff/models.py**

from django.contrib.auth.models import AbstractUser

from django.db import models

class User(AbstractUser):

    ROLE\_CHOICES = [

        ('trainee', 'Trainee'),

        ('team\_member', 'Team Member'),

        ('team\_leader', 'Team Leader'),

        ('manager', 'Manager'),

    ]

    role = models.CharField(max\_length=20, choices=ROLE\_CHOICES)

    def \_\_str\_\_(self):

        return f"{self.username} ({self.get\_role\_display()})"

**Step 3: Configure Custom User Model**

**In settings.py:**

AUTH\_USER\_MODEL = 'staff.User'

**Step 4: Migrate & Create Superuser**

python manage.py makemigrations

python manage.py migrate

python manage.py createsuperuser

**Step 5: Serializers**

**staff/serializers.py**

from rest\_framework import serializers

from .models import User

class UserSerializer(serializers.ModelSerializer):

    password = serializers.CharField(write\_only=True)

    class Meta:

        model = User

        fields = ['id', 'username', 'email', 'password', 'role']

    def create(self, validated\_data):

        password = validated\_data.pop('password')

        user = User(\*\*validated\_data)

        user.set\_password(password)

        if validated\_data.pop('role') == 'trainee':

            user.is\_staff = False  # To allow admin login if needed

        else:

            user.is\_staff = True

        user.save()

        return user

**Step 6: Permissions**

**staff/permissions.py**

from rest\_framework.permissions import BasePermission

class IsManager(BasePermission):

    def has\_permission(self, request, view):

        return request.user.is\_authenticated and request.user.role == 'manager'

class IsTeamLeaderOrManager(BasePermission):

    def has\_permission(self, request, view):

        return request.user.is\_authenticated and request.user.role in ['manager', 'team\_leader']

**Step 7: Views with CRUD**

**staff/views.py**

from rest\_framework import viewsets

from rest\_framework import generics, permissions

from rest\_framework.permissions import IsAuthenticated

from .models import User

from .serializers import UserSerializer

from .permissions import IsManager, IsTeamLeaderOrManager

class UserViewSet(viewsets.ModelViewSet):

    queryset = User.objects.all()

    serializer\_class = UserSerializer

    def get\_permissions(self):

        if self.action in ['create', 'destroy']:

            permission\_classes = [IsManager]

        elif self.action in ['update', 'partial\_update']:

            permission\_classes = [IsTeamLeaderOrManager]

        else:

            permission\_classes = [IsAuthenticated]

        return [permission() for permission in permission\_classes]

class UserCreateAPIView(generics.CreateAPIView):

    queryset = User.objects.all()

    serializer\_class = UserSerializer

    permission\_classes = [permissions.AllowAny]  # Allow anyone to create users (change if needed)

**Step 8: URLs**

**staff/urls.py**

from rest\_framework.routers import DefaultRouter

from .views import UserViewSet

from django.urls import path

from .views import UserCreateAPIView

router = DefaultRouter()

router.register(r'users', UserViewSet)

urlpatterns = router.urls

# from django.urls import path

# from .views import UserCreateAPIView

urlpatterns = [

    path('api/users/', UserCreateAPIView.as\_view(), name='user-create'),

]

**CompanyProject/urls.py**

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('api/', include('staff.urls')),

    path('', include('staff.urls')),

]

**Step 9: Authentication Setup**

You can use DRF's built-in session authentication or token authentication.

**Enable Session Auth in settings.py**

REST\_FRAMEWORK = {

    'DEFAULT\_AUTHENTICATION\_CLASSES': [

        'rest\_framework.authentication.SessionAuthentication',

    ],

    'DEFAULT\_PERMISSION\_CLASSES': [

        'rest\_framework.permissions.IsAuthenticated',

    ],

}

**Step 10: Test**

python manage.py runserver

**Add Fixtures for Test Users**

**Start Shell:**

python manage.py shell

**Insert Full code:**

**Insert Test Users:**

from staff.models import User

User.objects.create\_user(username="trainee1", email="trainee1@example.com", password="pass123", role="trainee")

User.objects.create\_user(username="member1", email="member1@example.com", password="pass123", role="team\_member")

User.objects.create\_user(username="leader1", email="leader1@example.com", password="pass123", role="team\_leader")

User.objects.create\_user(username="manager1", email="manager1@example.com", password="pass123", role="manager")

**Assign Test Users Staff role:**

from staff.models import User

User.objects.filter(username="trainee1").update(is\_staff=False)

User.objects.filter(username="member1").update(is\_staff=True)

User.objects.filter(username="leader1").update(is\_staff=True)

User.objects.filter(username="manager1").update(is\_staff=True)

**Exit Shell:**

exit()

 **Django admin panel:** <http://127.0.0.1:8000/admin>

 **DRF API login:** <http://127.0.0.1:8000/api/users/>

**Show Custom User Model in Admin**

**🔹 Step 1: Open staff/admin.py**

from django.contrib import admin

from django.contrib.auth.admin import UserAdmin as BaseUserAdmin

from django.contrib.auth.models import Permission

from django.contrib.contenttypes.models import ContentType

from .models import User

class UserAdmin(BaseUserAdmin):

    list\_display = ('username', 'email', 'role', 'is\_staff', 'is\_superuser')

    list\_filter = ('role', 'is\_staff', 'is\_superuser')

    fieldsets = (

        (None, {'fields': ('username', 'password')}),

        ('Personal Info', {'fields': ('first\_name', 'last\_name', 'email')}),

        ('Permissions', {'fields': ('role', 'is\_active', 'is\_staff', 'is\_superuser', 'groups', 'user\_permissions')}),

    )

    add\_fieldsets = (

        (None, {

            'classes': ('wide',),

            'fields': ('username', 'email', 'role', 'password1', 'password2'),

        }),

    )

    search\_fields = ('username', 'email')

    ordering = ('username',)

    def get\_queryset(self, request):

        qs = super().get\_queryset(request)

        if request.user.is\_superuser:

            return qs  # Superuser can see everyone

        # Exclude superusers for all non-superusers

        qs = qs.exclude(is\_superuser=True)

        if request.user.role == 'manager':

            return qs

        elif request.user.role == 'team\_leader':

            return qs.filter(role\_\_in=['team\_member', 'trainee'])

        else:

            return qs.filter(id=request.user.id)

    # Role-based view permission

    def has\_view\_permission(self, request, obj=None):

        if request.user.is\_superuser or request.user.role == 'manager':

            return True

        if obj is None:

            return True

        if request.user.role == 'team\_leader':

            return obj.role in ['team\_member', 'trainee']

        return obj.id == request.user.id

    # Role-based change permission

    def has\_change\_permission(self, request, obj=None):

        if request.user.is\_superuser or request.user.role == 'manager':

            return True

        if obj is None:

            return False

        return obj.id == request.user.id

    # Role-based delete permission

    def has\_delete\_permission(self, request, obj=None):

        return request.user.is\_superuser or request.user.role == 'manager'

    # Role-based add permission

    def has\_add\_permission(self, request):

        return request.user.is\_superuser or request.user.role == 'manager'

    # Make sensitive fields read-only for non-managers

    def get\_readonly\_fields(self, request, obj=None):

        readonly\_fields = super().get\_readonly\_fields(request, obj)

        if not request.user.is\_superuser and request.user.role != 'manager':

            return readonly\_fields + ('role', 'is\_staff', 'is\_superuser')

        return readonly\_fields

    # Automatically assign permissions when saving user

    def save\_model(self, request, obj, form, change):

        super().save\_model(request, obj, form, change)

        # Clear all existing permissions

        obj.user\_permissions.clear()

        # Get user model content type

        try:

            content\_type = ContentType.objects.get(app\_label='staff', model='user')

        except ContentType.DoesNotExist:

            return

        # Assign permissions based on role

        if obj.role == 'manager':

            perms = ['view\_user', 'add\_user', 'change\_user', 'delete\_user']

        elif obj.role == 'team\_leader':

            perms = ['view\_user', 'add\_user']

        elif obj.role == 'team\_member':

            perms = ['view\_user']

        else:

            perms = []

        # Grant permissions

        permissions = Permission.objects.filter(codename\_\_in=perms, content\_type=content\_type)

        obj.user\_permissions.set(permissions)

        # Allow admin access only if they have any permission

        obj.is\_staff = bool(perms)

        obj.save()

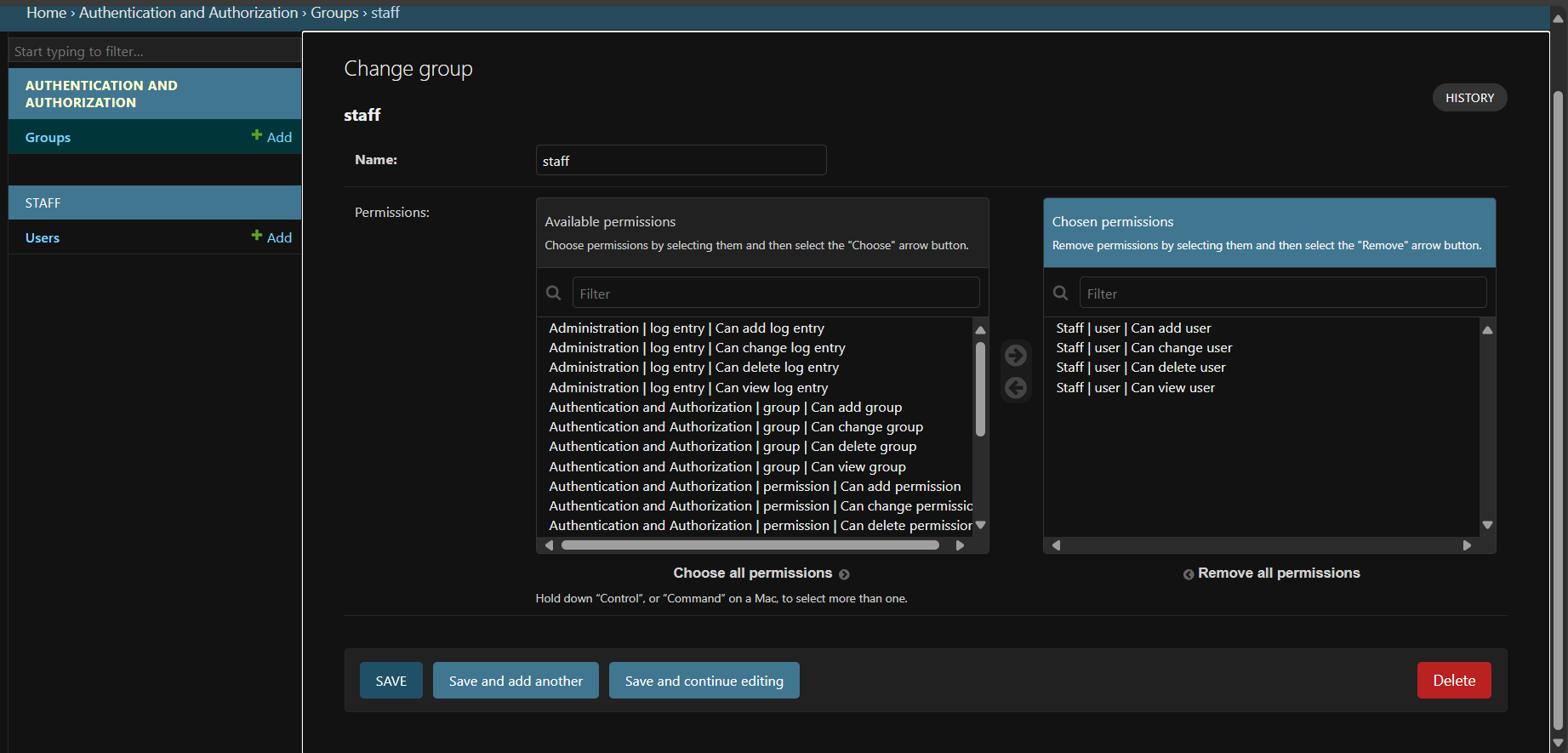
# Register the user model with the custom admin

admin.site.register(User, UserAdmin)

**🔹 Step 2: Restart your development server**

python manage.py runserver

**Make a group staff in admin panel and add following permissions**

****

**Restart Server(if changes don’t appear)**

**Create staff/signals.py**

**Create a signal to add staff users to the group**

from django.db.models.signals import post\_save

from django.dispatch import receiver

from django.contrib.auth.models import Group

from .models import User

@receiver(post\_save, sender=User)

def add\_user\_to\_staff\_group(sender, instance, created, \*\*kwargs):

    # Define staff roles

    staff\_roles = ['manager', 'team\_leader', 'team\_member', 'trainee']

    if instance.role in staff\_roles:

        # Get or create the 'staff' group

        staff\_group, \_ = Group.objects.get\_or\_create(name='staff')

        # Add user to the 'staff' group if not already in it

        if not instance.groups.filter(name='staff').exists():

            instance.groups.add(staff\_group)

**Why use signals?**

* **The permission assignment logic runs every time a user is created or updated, anywhere in your app.**
* **Keeps admin code clean and separates concerns.**
* **Works well with APIs, shell scripts, and admin.**

**Connect to staff/apps.py**

from django.apps import AppConfig

class StaffConfig(AppConfig):

    default\_auto\_field = 'django.db.models.BigAutoField'

    name = 'staff'

    def ready(self):

        import staff.signals  # noqa

**Create users with roles**

**Folder Structure**

**CompanyProject/**

**│**

**├── CompanyProject/ # Main project directory**

**│ ├── \_\_init\_\_.py**

**│ ├── settings.py # Project settings**

**│ ├── urls.py # URL routing for the project**

**│ ├── wsgi.py # WSGI configuration**

**│ └── asgi.py # ASGI configuration**

**│**

**├── staff/ # Django app for user management**

**│ ├── migrations/ # Database migrations**

**│ │ └── \_\_init\_\_.py**

**│ ├── \_\_init\_\_.py**

**│ ├── admin.py # Admin interface customization**

**│ ├── apps.py # App configuration**

**│ ├── models.py # Data models (including custom user model)**

**│ ├── serializers.py # Serializers for API**

**│ ├── tests.py # Unit tests**

**│ ├── urls.py # URL routing for the staff app**

**│ ├── views.py # API views for user management**

**│ └── permissions.py # Custom permissions**

**│**

**├── db.sqlite3 # SQLite database file (if using SQLite)**

**├── manage.py # Command-line utility for administrative tasks**

**└── requirements.txt # List of project dependencies (optional)**

**Thank You**

* **Aditya Bhardwaj**