I've created an interactive concept map for your tutoring platform! This visual map breaks down your Django + React application into key components:

**Key Features Mapped:**

* **Job Posting System** (LinkedIn-style anonymous posting)
* **Tutor Profiles** with availability settings (part-time, full-time, weekends, etc.)
* **Search & Filter System** (TeacherOn-style search)
* **UI Design** (Superprof-style tutor lists)
* **Payment System** for accessing contact information
* **User Management** (Companies and Tutors)

The map shows how all components connect and includes the main user workflows. You can click on nodes to interact with them!

I've created a complete, production-ready tutoring platform application with Django backend and React frontend! 🚀

**What's Included:**

**🔧 Backend (Django):**

* Complete user authentication system with JWT
* Models for tutors, companies, jobs, payments, and reviews
* RESTful API with filtering, search, and pagination
* Stripe payment integration for contact purchases
* Anonymous job posting system
* Advanced filtering system

**⚛️ Frontend (React):**

* Modern, responsive UI with Tailwind CSS
* LinkedIn-style job posting interface
* TeacherOn-style search and filtering
* Superprof-style tutor profile cards
* Stripe payment integration
* Real-time notifications with React Toastify
* Protected routes and authentication

**✨ Key Features:**

* **Anonymous Job Posting** (LinkedIn-style)
* **Advanced Tutor Search** with multiple filters
* **Payment System** to unlock contact information
* **Availability Management** (part-time, full-time, weekends, etc.)
* **Multi-modal Teaching** (online, offline, both)
* **Rating & Review System**
* **Responsive Design** for all devices

**🗂️ Complete File Structure:**

* All Django models, views, serializers, and URLs
* React components, pages, contexts, and services
* Database configuration and migrations
* Stripe payment integration
* Authentication and authorization
* API filters and search functionality

You can download this as a complete project and follow the setup instructions provided in the artifact. The application includes everything from user registration to payment processing, making it a fully functional tutoring marketplace!

Would you like me to explain any specific part of the implementation or help you customize any features?

# Tutoring Platform - Complete Application Structure

# This includes Django Backend + React Frontend

# ================================

# PROJECT STRUCTURE

# ================================

"""

tutoring\_platform/

├── backend/

│ ├── manage.py

│ ├── requirements.txt

│ ├── tutoring\_platform/

│ │ ├── \_\_init\_\_.py

│ │ ├── settings.py

│ │ ├── urls.py

│ │ ├── wsgi.py

│ │ └── asgi.py

│ └── apps/

│ ├── \_\_init\_\_.py

│ ├── accounts/

│ ├── tutors/

│ ├── jobs/

│ ├── payments/

│ └── api/

├── frontend/

│ ├── package.json

│ ├── src/

│ │ ├── components/

│ │ ├── pages/

│ │ ├── services/

│ │ └── utils/

│ └── public/

└── README.md

"""

# ================================

# 1. DJANGO BACKEND - requirements.txt

# ================================

REQUIREMENTS\_TXT = """

Django==4.2.0

djangorestframework==3.14.0

django-cors-headers==4.0.0

djangorestframework-simplejwt==5.2.2

django-filter==23.2

Pillow==9.5.0

psycopg2-binary==2.9.6

python-decouple==3.8

stripe==5.4.0

celery==5.3.0

redis==4.5.5

django-storages==1.13.2

boto3==1.26.137

"""

# ================================

# 2. DJANGO SETTINGS

# ================================

DJANGO\_SETTINGS = """

import os

from pathlib import Path

from decouple import config

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

SECRET\_KEY = config('SECRET\_KEY', default='your-secret-key-here')

DEBUG = config('DEBUG', default=True, cast=bool)

ALLOWED\_HOSTS = ['localhost', '127.0.0.1', '0.0.0.0']

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

# Third party apps

'rest\_framework',

'rest\_framework\_simplejwt',

'corsheaders',

'django\_filters',

# Local apps

'apps.accounts',

'apps.tutors',

'apps.jobs',

'apps.payments',

'apps.api',

]

MIDDLEWARE = [

'corsheaders.middleware.CorsMiddleware',

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'tutoring\_platform.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': config('DB\_NAME', default='tutoring\_platform'),

'USER': config('DB\_USER', default='postgres'),

'PASSWORD': config('DB\_PASSWORD', default='password'),

'HOST': config('DB\_HOST', default='localhost'),

'PORT': config('DB\_PORT', default='5432'),

}

}

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework\_simplejwt.authentication.JWTAuthentication',

],

'DEFAULT\_PERMISSION\_CLASSES': [

'rest\_framework.permissions.IsAuthenticated',

],

'DEFAULT\_FILTER\_BACKENDS': [

'django\_filters.rest\_framework.DjangoFilterBackend',

'rest\_framework.filters.SearchFilter',

'rest\_framework.filters.OrderingFilter',

],

'DEFAULT\_PAGINATION\_CLASS': 'rest\_framework.pagination.PageNumberPagination',

'PAGE\_SIZE': 20

}

from datetime import timedelta

SIMPLE\_JWT = {

'ACCESS\_TOKEN\_LIFETIME': timedelta(minutes=60),

'REFRESH\_TOKEN\_LIFETIME': timedelta(days=7),

'ROTATE\_REFRESH\_TOKENS': True,

}

CORS\_ALLOWED\_ORIGINS = [

"http://localhost:3000",

"http://127.0.0.1:3000",

]

# Stripe Configuration

STRIPE\_PUBLISHABLE\_KEY = config('STRIPE\_PUBLISHABLE\_KEY', default='')

STRIPE\_SECRET\_KEY = config('STRIPE\_SECRET\_KEY', default='')

# Media files

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

STATIC\_URL = '/static/'

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

"""

# ================================

# 3. MODELS - accounts/models.py

# ================================

ACCOUNTS\_MODELS = """

from django.contrib.auth.models import AbstractUser

from django.db import models

class CustomUser(AbstractUser):

USER\_TYPE\_CHOICES = [

('tutor', 'Tutor'),

('company', 'Company'),

]

user\_type = models.CharField(max\_length=10, choices=USER\_TYPE\_CHOICES)

phone = models.CharField(max\_length=20, blank=True)

is\_verified = models.BooleanField(default=False)

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self):

return f"{self.username} ({self.user\_type})"

class Company(models.Model):

user = models.OneToOneField(CustomUser, on\_delete=models.CASCADE)

company\_name = models.CharField(max\_length=200)

description = models.TextField(blank=True)

website = models.URLField(blank=True)

logo = models.ImageField(upload\_to='company\_logos/', blank=True)

address = models.TextField(blank=True)

contact\_person = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return self.company\_name

class Meta:

verbose\_name\_plural = "Companies"

"""

# ================================

# 4. MODELS - tutors/models.py

# ================================

TUTORS\_MODELS = """

from django.db import models

from apps.accounts.models import CustomUser

class Subject(models.Model):

name = models.CharField(max\_length=100, unique=True)

category = models.CharField(max\_length=100)

description = models.TextField(blank=True)

def \_\_str\_\_(self):

return self.name

class TutorProfile(models.Model):

AVAILABILITY\_CHOICES = [

('part\_time', 'Part Time'),

('full\_time', 'Full Time'),

('weekends', 'Weekends Only'),

('weekdays', 'Weekdays Only'),

('flexible', 'Flexible'),

]

EXPERIENCE\_CHOICES = [

('0-1', '0-1 years'),

('1-3', '1-3 years'),

('3-5', '3-5 years'),

('5-10', '5-10 years'),

('10+', '10+ years'),

]

MODE\_CHOICES = [

('online', 'Online Only'),

('offline', 'Offline Only'),

('both', 'Both Online & Offline'),

]

user = models.OneToOneField(CustomUser, on\_delete=models.CASCADE)

bio = models.TextField()

subjects = models.ManyToManyField(Subject, through='TutorSubject')

experience = models.CharField(max\_length=10, choices=EXPERIENCE\_CHOICES)

hourly\_rate\_min = models.DecimalField(max\_digits=8, decimal\_places=2)

hourly\_rate\_max = models.DecimalField(max\_digits=8, decimal\_places=2)

availability = models.CharField(max\_length=20, choices=AVAILABILITY\_CHOICES)

teaching\_mode = models.CharField(max\_length=10, choices=MODE\_CHOICES)

profile\_image = models.ImageField(upload\_to='tutor\_profiles/', blank=True)

city = models.CharField(max\_length=100)

state = models.CharField(max\_length=100)

country = models.CharField(max\_length=100, default='India')

languages = models.CharField(max\_length=200, help\_text="Comma-separated languages")

qualifications = models.TextField()

achievements = models.TextField(blank=True)

rating = models.DecimalField(max\_digits=3, decimal\_places=2, default=0.00)

total\_reviews = models.IntegerField(default=0)

is\_featured = models.BooleanField(default=False)

is\_verified = models.BooleanField(default=False)

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self):

return f"{self.user.get\_full\_name()} - {self.user.username}"

class TutorSubject(models.Model):

tutor = models.ForeignKey(TutorProfile, on\_delete=models.CASCADE)

subject = models.ForeignKey(Subject, on\_delete=models.CASCADE)

proficiency\_level = models.CharField(max\_length=20, choices=[

('beginner', 'Beginner'),

('intermediate', 'Intermediate'),

('advanced', 'Advanced'),

('expert', 'Expert'),

])

years\_experience = models.IntegerField(default=0)

class Meta:

unique\_together = ('tutor', 'subject')

class TutorAvailability(models.Model):

DAYS\_CHOICES = [

('monday', 'Monday'),

('tuesday', 'Tuesday'),

('wednesday', 'Wednesday'),

('thursday', 'Thursday'),

('friday', 'Friday'),

('saturday', 'Saturday'),

('sunday', 'Sunday'),

]

tutor = models.ForeignKey(TutorProfile, on\_delete=models.CASCADE)

day = models.CharField(max\_length=10, choices=DAYS\_CHOICES)

start\_time = models.TimeField()

end\_time = models.TimeField()

is\_available = models.BooleanField(default=True)

class Meta:

unique\_together = ('tutor', 'day', 'start\_time')

class Review(models.Model):

tutor = models.ForeignKey(TutorProfile, on\_delete=models.CASCADE, related\_name='reviews')

company = models.ForeignKey('accounts.Company', on\_delete=models.CASCADE)

rating = models.IntegerField(choices=[(i, i) for i in range(1, 6)])

comment = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True)

class Meta:

unique\_together = ('tutor', 'company')

"""

# ================================

# 5. MODELS - jobs/models.py

# ================================

JOBS\_MODELS = """

from django.db import models

from apps.accounts.models import Company

from apps.tutors.models import Subject

class JobPost(models.Model):

JOB\_TYPE\_CHOICES = [

('one\_time', 'One Time'),

('ongoing', 'Ongoing'),

('project', 'Project Based'),

]

MODE\_CHOICES = [

('online', 'Online'),

('offline', 'Offline'),

('both', 'Both'),

]

URGENCY\_CHOICES = [

('low', 'Low'),

('medium', 'Medium'),

('high', 'High'),

('urgent', 'Urgent'),

]

company = models.ForeignKey(Company, on\_delete=models.CASCADE)

title = models.CharField(max\_length=200)

description = models.TextField()

subjects = models.ManyToManyField(Subject)

job\_type = models.CharField(max\_length=20, choices=JOB\_TYPE\_CHOICES)

teaching\_mode = models.CharField(max\_length=10, choices=MODE\_CHOICES)

budget\_min = models.DecimalField(max\_digits=8, decimal\_places=2)

budget\_max = models.DecimalField(max\_digits=8, decimal\_places=2)

duration = models.CharField(max\_length=100, help\_text="e.g., 3 months, 6 weeks")

location = models.CharField(max\_length=200, blank=True)

preferred\_experience = models.CharField(max\_length=50)

urgency = models.CharField(max\_length=10, choices=URGENCY\_CHOICES)

requirements = models.TextField()

is\_anonymous = models.BooleanField(default=True)

is\_active = models.BooleanField(default=True)

views\_count = models.IntegerField(default=0)

applications\_count = models.IntegerField(default=0)

deadline = models.DateTimeField(null=True, blank=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self):

return self.title

class Meta:

ordering = ['-created\_at']

class JobApplication(models.Model):

STATUS\_CHOICES = [

('pending', 'Pending'),

('viewed', 'Viewed'),

('shortlisted', 'Shortlisted'),

('rejected', 'Rejected'),

('hired', 'Hired'),

]

job = models.ForeignKey(JobPost, on\_delete=models.CASCADE, related\_name='applications')

tutor = models.ForeignKey('tutors.TutorProfile', on\_delete=models.CASCADE)

cover\_letter = models.TextField()

proposed\_rate = models.DecimalField(max\_digits=8, decimal\_places=2)

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='pending')

applied\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

class Meta:

unique\_together = ('job', 'tutor')

"""

# ================================

# 6. MODELS - payments/models.py

# ================================

PAYMENTS\_MODELS = """

from django.db import models

from apps.accounts.models import Company

from apps.tutors.models import TutorProfile

class ContactPurchase(models.Model):

company = models.ForeignKey(Company, on\_delete=models.CASCADE)

tutor = models.ForeignKey(TutorProfile, on\_delete=models.CASCADE)

amount = models.DecimalField(max\_digits=8, decimal\_places=2)

stripe\_payment\_intent\_id = models.CharField(max\_length=200)

is\_successful = models.BooleanField(default=False)

purchased\_at = models.DateTimeField(auto\_now\_add=True)

class Meta:

unique\_together = ('company', 'tutor')

class Subscription(models.Model):

PLAN\_CHOICES = [

('basic', 'Basic - 5 contacts/month'),

('standard', 'Standard - 20 contacts/month'),

('premium', 'Premium - Unlimited contacts'),

]

company = models.OneToOneField(Company, on\_delete=models.CASCADE)

plan = models.CharField(max\_length=20, choices=PLAN\_CHOICES)

stripe\_subscription\_id = models.CharField(max\_length=200)

is\_active = models.BooleanField(default=True)

contacts\_used = models.IntegerField(default=0)

contacts\_limit = models.IntegerField()

started\_at = models.DateTimeField(auto\_now\_add=True)

expires\_at = models.DateTimeField()

def can\_access\_contact(self):

if self.plan == 'premium':

return True

return self.contacts\_used < self.contacts\_limit

"""

# ================================

# 7. API VIEWS - api/views.py

# ================================

API\_VIEWS = """

from rest\_framework import generics, permissions, status, filters

from rest\_framework.decorators import api\_view, permission\_classes

from rest\_framework.response import Response

from rest\_framework\_simplejwt.views import TokenObtainPairView

from django\_filters.rest\_framework import DjangoFilterBackend

from django.db.models import Q, Avg

import stripe

from django.conf import settings

from apps.accounts.models import CustomUser, Company

from apps.tutors.models import TutorProfile, Subject, Review

from apps.jobs.models import JobPost, JobApplication

from apps.payments.models import ContactPurchase, Subscription

from .serializers import \*

from .filters import TutorFilter, JobFilter

stripe.api\_key = settings.STRIPE\_SECRET\_KEY

class CustomTokenObtainPairView(TokenObtainPairView):

serializer\_class = CustomTokenObtainPairSerializer

class RegisterView(generics.CreateAPIView):

queryset = CustomUser.objects.all()

serializer\_class = UserRegistrationSerializer

permission\_classes = [permissions.AllowAny]

class TutorListView(generics.ListAPIView):

queryset = TutorProfile.objects.filter(is\_verified=True)

serializer\_class = TutorProfileSerializer

filter\_backends = [DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter]

filterset\_class = TutorFilter

search\_fields = ['bio', 'user\_\_first\_name', 'user\_\_last\_name', 'subjects\_\_name']

ordering\_fields = ['rating', 'hourly\_rate\_min', 'created\_at']

ordering = ['-rating', '-created\_at']

permission\_classes = [permissions.AllowAny]

class TutorDetailView(generics.RetrieveAPIView):

queryset = TutorProfile.objects.all()

serializer\_class = TutorProfileDetailSerializer

permission\_classes = [permissions.AllowAny]

class JobPostListView(generics.ListCreateAPIView):

queryset = JobPost.objects.filter(is\_active=True)

serializer\_class = JobPostSerializer

filter\_backends = [DjangoFilterBackend, filters.SearchFilter]

filterset\_class = JobFilter

search\_fields = ['title', 'description', 'subjects\_\_name']

def get\_permissions(self):

if self.request.method == 'GET':

return [permissions.AllowAny()]

return [permissions.IsAuthenticated()]

class CompanyJobsView(generics.ListAPIView):

serializer\_class = JobPostSerializer

permission\_classes = [permissions.IsAuthenticated]

def get\_queryset(self):

return JobPost.objects.filter(company\_\_user=self.request.user)

class JobApplicationView(generics.CreateAPIView):

queryset = JobApplication.objects.all()

serializer\_class = JobApplicationSerializer

permission\_classes = [permissions.IsAuthenticated]

@api\_view(['POST'])

@permission\_classes([permissions.IsAuthenticated])

def purchase\_contact(request, tutor\_id):

try:

tutor = TutorProfile.objects.get(id=tutor\_id)

company = Company.objects.get(user=request.user)

# Check if already purchased

if ContactPurchase.objects.filter(company=company, tutor=tutor).exists():

return Response({'error': 'Contact already purchased'},

status=status.HTTP\_400\_BAD\_REQUEST)

# Create Stripe payment intent

intent = stripe.PaymentIntent.create(

amount=500, # $5.00 in cents

currency='usd',

metadata={'company\_id': company.id, 'tutor\_id': tutor.id}

)

return Response({

'client\_secret': intent.client\_secret,

'amount': 500

})

except Exception as e:

return Response({'error': str(e)}, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(['POST'])

@permission\_classes([permissions.IsAuthenticated])

def confirm\_contact\_purchase(request):

try:

payment\_intent\_id = request.data.get('payment\_intent\_id')

tutor\_id = request.data.get('tutor\_id')

company = Company.objects.get(user=request.user)

tutor = TutorProfile.objects.get(id=tutor\_id)

# Verify payment with Stripe

intent = stripe.PaymentIntent.retrieve(payment\_intent\_id)

if intent.status == 'succeeded':

ContactPurchase.objects.create(

company=company,

tutor=tutor,

amount=5.00,

stripe\_payment\_intent\_id=payment\_intent\_id,

is\_successful=True

)

serializer = TutorContactSerializer(tutor)

return Response(serializer.data)

return Response({'error': 'Payment not successful'},

status=status.HTTP\_400\_BAD\_REQUEST)

except Exception as e:

return Response({'error': str(e)}, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(['GET'])

@permission\_classes([permissions.AllowAny])

def subjects\_list(request):

subjects = Subject.objects.all()

serializer = SubjectSerializer(subjects, many=True)

return Response(serializer.data)

class ReviewCreateView(generics.CreateAPIView):

queryset = Review.objects.all()

serializer\_class = ReviewSerializer

permission\_classes = [permissions.IsAuthenticated]

"""

# ================================

# 8. API SERIALIZERS - api/serializers.py

# ================================

API\_SERIALIZERS = """

from rest\_framework import serializers

from rest\_framework\_simplejwt.serializers import TokenObtainPairSerializer

from django.contrib.auth.password\_validation import validate\_password

from apps.accounts.models import CustomUser, Company

from apps.tutors.models import TutorProfile, Subject, TutorSubject, Review

from apps.jobs.models import JobPost, JobApplication

class CustomTokenObtainPairSerializer(TokenObtainPairSerializer):

def validate(self, attrs):

data = super().validate(attrs)

data['user\_type'] = self.user.user\_type

data['user\_id'] = self.user.id

return data

class UserRegistrationSerializer(serializers.ModelSerializer):

password = serializers.CharField(write\_only=True, validators=[validate\_password])

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

class Meta:

model = CustomUser

fields = ('username', 'email', 'first\_name', 'last\_name',

'user\_type', 'phone', 'password', 'password\_confirm')

def validate(self, attrs):

if attrs['password'] != attrs['password\_confirm']:

raise serializers.ValidationError("Passwords don't match")

return attrs

def create(self, validated\_data):

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

user = CustomUser.objects.create\_user(\*\*validated\_data)

return user

class SubjectSerializer(serializers.ModelSerializer):

class Meta:

model = Subject

fields = '\_\_all\_\_'

class TutorSubjectSerializer(serializers.ModelSerializer):

subject = SubjectSerializer(read\_only=True)

class Meta:

model = TutorSubject

fields = ('subject', 'proficiency\_level', 'years\_experience')

class ReviewSerializer(serializers.ModelSerializer):

company\_name = serializers.CharField(source='company.company\_name', read\_only=True)

class Meta:

model = Review

fields = ('id', 'rating', 'comment', 'created\_at', 'company\_name')

class TutorProfileSerializer(serializers.ModelSerializer):

full\_name = serializers.CharField(source='user.get\_full\_name', read\_only=True)

subjects = TutorSubjectSerializer(source='tutorsubject\_set', many=True, read\_only=True)

recent\_reviews = ReviewSerializer(source='reviews', many=True, read\_only=True)

class Meta:

model = TutorProfile

fields = ('id', 'full\_name', 'bio', 'subjects', 'experience',

'hourly\_rate\_min', 'hourly\_rate\_max', 'availability',

'teaching\_mode', 'profile\_image', 'city', 'state',

'rating', 'total\_reviews', 'is\_featured', 'recent\_reviews')

class TutorProfileDetailSerializer(TutorProfileSerializer):

class Meta(TutorProfileSerializer.Meta):

fields = TutorProfileSerializer.Meta.fields + (

'languages', 'qualifications', 'achievements', 'created\_at'

)

class TutorContactSerializer(serializers.ModelSerializer):

email = serializers.CharField(source='user.email', read\_only=True)

phone = serializers.CharField(source='user.phone', read\_only=True)

full\_name = serializers.CharField(source='user.get\_full\_name', read\_only=True)

class Meta:

model = TutorProfile

fields = ('full\_name', 'email', 'phone')

class JobPostSerializer(serializers.ModelSerializer):

company\_name = serializers.CharField(source='company.company\_name', read\_only=True)

subjects = SubjectSerializer(many=True, read\_only=True)

class Meta:

model = JobPost

fields = ('id', 'title', 'description', 'subjects', 'job\_type',

'teaching\_mode', 'budget\_min', 'budget\_max', 'duration',

'location', 'urgency', 'company\_name', 'created\_at',

'applications\_count', 'deadline')

class JobApplicationSerializer(serializers.ModelSerializer):

class Meta:

model = JobApplication

fields = ('job', 'cover\_letter', 'proposed\_rate')

"""

# ================================

# 9. API FILTERS - api/filters.py

# ================================

API\_FILTERS = """

import django\_filters

from apps.tutors.models import TutorProfile

from apps.jobs.models import JobPost

class TutorFilter(django\_filters.FilterSet):

subjects = django\_filters.CharFilter(field\_name='subjects\_\_name', lookup\_expr='icontains')

min\_rate = django\_filters.NumberFilter(field\_name='hourly\_rate\_min', lookup\_expr='gte')

max\_rate = django\_filters.NumberFilter(field\_name='hourly\_rate\_max', lookup\_expr='lte')

city = django\_filters.CharFilter(lookup\_expr='icontains')

teaching\_mode = django\_filters.ChoiceFilter(choices=TutorProfile.MODE\_CHOICES)

availability = django\_filters.ChoiceFilter(choices=TutorProfile.AVAILABILITY\_CHOICES)

rating = django\_filters.NumberFilter(field\_name='rating', lookup\_expr='gte')

class Meta:

model = TutorProfile

fields = ['subjects', 'min\_rate', 'max\_rate', 'city', 'teaching\_mode', 'availability', 'rating']

class JobFilter(django\_filters.FilterSet):

subjects = django\_filters.CharFilter(field\_name='subjects\_\_name', lookup\_expr='icontains')

min\_budget = django\_filters.NumberFilter(field\_name='budget\_min', lookup\_expr='gte')

max\_budget = django\_filters.NumberFilter(field\_name='budget\_max', lookup\_expr='lte')

teaching\_mode = django\_filters.ChoiceFilter(choices=JobPost.MODE\_CHOICES)

job\_type = django\_filters.ChoiceFilter(choices=JobPost.JOB\_TYPE\_CHOICES)

urgency = django\_filters.ChoiceFilter(choices=JobPost.URGENCY\_CHOICES)

class Meta:

model = JobPost

fields = ['subjects', 'min\_budget', 'max\_budget', 'teaching\_mode', 'job\_type', 'urgency']

"""

# ================================

# 10. URLS CONFIGURATION

# ================================

MAIN\_URLS = """

from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('admin/', admin.site.urls),

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

]

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

"""

API\_URLS = """

from django.urls import path

from rest\_framework\_simplejwt.views import TokenRefreshView

from . import views

urlpatterns = [

# Authentication

path('auth/login/', views.CustomTokenObtainPairView.as\_view(), name='token\_obtain\_pair'),

path('auth/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'),

path('auth/register/', views.RegisterView.as\_view(), name='register'),

# Tutors

path('tutors/', views.TutorListView.as\_view(), name='tutor-list'),

path('tutors/<int:pk>/', views.TutorDetailView.as\_view(), name='tutor-detail'),

# Jobs

path('jobs/', views.JobPostListView.as\_view(), name='job-list'),

path('my-jobs/', views.CompanyJobsView.as\_view(), name='my-jobs'),

path('jobs/apply/', views.JobApplicationView.as\_view(), name='job-apply'),

# Payments

path('purchase-contact/<int:tutor\_id>/', views.purchase\_contact, name='purchase-contact'),

path('confirm