Here is the full implementation with all the updates, including KYC document upload and PostgreSQL integration.

**Step 1: Create Virtual Environment and Install Dependencies**

Run the following commands in your terminal:

bash

Copy code

# Install virtualenv if you don't have it already

pip install virtualenv

# Create a virtual environment

virtualenv env

# Activate the virtual environment

source env/bin/activate # For Windows, use `env\Scripts\activate`

# Install Django and other dependencies

pip install django djangorestframework djangorestframework-simplejwt psycopg2

# Save dependencies to requirements.txt

pip freeze > requirements.txt

**Step 2: Create Django Project and User Management App**

bash

Copy code

# Create Django project and app

django-admin startproject stock\_simulation

cd stock\_simulation

python manage.py startapp users

**Step 3: Configure PostgreSQL in Django**

1. **Create PostgreSQL Database:**

Create a new PostgreSQL database and user by opening psql:

sql

Copy code

CREATE DATABASE stock\_simulation\_db;

CREATE USER stock\_user WITH PASSWORD 'your\_password';

ALTER ROLE stock\_user SET client\_encoding TO 'utf8';

ALTER ROLE stock\_user SET default\_transaction\_isolation TO 'read committed';

ALTER ROLE stock\_user SET timezone TO 'UTC';

GRANT ALL PRIVILEGES ON DATABASE stock\_simulation\_db TO stock\_user;

1. **Update Django settings.py for PostgreSQL:**

In stock\_simulation/settings.py, configure the database to use PostgreSQL:

python

Copy code

DATABASES = {

'default': {

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

'NAME': 'stock\_simulation\_db',

'USER': 'stock\_user',

'PASSWORD': 'your\_password',

'HOST': 'localhost',

'PORT': '5432',

}

}

Also, configure Django to handle media files (for KYC document uploads):

python

Copy code

import os

MEDIA\_URL = '/media/'

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

**Step 4: Create Custom User Model with KYC Fields**

In users/models.py, create the custom CustomUser model with KYC fields:

python

Copy code

from django.contrib.auth.models import AbstractUser

from django.db import models

class CustomUser(AbstractUser):

ROLE\_CHOICES = [

('trader', 'Trader'),

('regulator', 'Regulator'),

('company\_admin', 'Company Admin'),

]

role = models.CharField(max\_length=15, choices=ROLE\_CHOICES, default='trader')

# KYC fields

kyc\_document = models.FileField(upload\_to='kyc\_documents/', blank=True, null=True)

kyc\_verified = models.BooleanField(default=False) # Admin will verify this

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

return self.username

In stock\_simulation/settings.py, add the custom user model configuration:

python

Copy code

AUTH\_USER\_MODEL = 'users.CustomUser'

**Step 5: Set Up JWT Authentication**

1. **Add JWT Authentication to settings.py:**

In stock\_simulation/settings.py, add the following settings to configure JWT authentication and token expiration:

python

Copy code

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': (

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

),

}

from datetime import timedelta

SIMPLE\_JWT = {

'ACCESS\_TOKEN\_LIFETIME': timedelta(days=2), # Token expires in 2 days

'REFRESH\_TOKEN\_LIFETIME': timedelta(days=7), # Refresh token expires in 7 days

'ROTATE\_REFRESH\_TOKENS': True,

'BLACKLIST\_AFTER\_ROTATION': True,

}

1. **Create Serializer for User Registration and KYC:**

In users/serializers.py, define the UserSerializer to handle user registration and KYC:

python

Copy code

from django.contrib.auth import get\_user\_model

from rest\_framework import serializers

User = get\_user\_model()

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = ('username', 'email', 'password', 'role', 'kyc\_document', 'kyc\_verified')

extra\_kwargs = {'password': {'write\_only': True}, 'kyc\_verified': {'read\_only': True}}

def create(self, validated\_data):

user = User.objects.create\_user(

username=validated\_data['username'],

password=validated\_data['password'],

email=validated\_data['email'],

role=validated\_data['role'],

kyc\_document=validated\_data.get('kyc\_document', None),

)

return user

1. **Create Registration and Login Views:**

In users/views.py, create the view for user registration and authentication:

python

Copy code

from django.contrib.auth import get\_user\_model

from rest\_framework import generics

from rest\_framework.permissions import AllowAny

from rest\_framework.parsers import MultiPartParser, FormParser

from .serializers import UserSerializer

User = get\_user\_model()

class RegisterUser(generics.CreateAPIView):

queryset = User.objects.all()

permission\_classes = (AllowAny,)

serializer\_class = UserSerializer

parser\_classes = (MultiPartParser, FormParser) # For file uploads

1. **Set up URLs for User Management:**

In users/urls.py, define routes for registration, login, and token refresh:

python

Copy code

from django.urls import path

from rest\_framework\_simplejwt.views import (

TokenObtainPairView,

TokenRefreshView,

)

from .views import RegisterUser

urlpatterns = [

path('register/', RegisterUser.as\_view(), name='register'),

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

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

]

Then include these routes in the main project urls.py:

python

Copy code

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

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

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

**Step 6: Run Migrations and Start Django Server**

1. **Apply Migrations:**

Run the following commands to apply the database migrations for PostgreSQL:

bash

Copy code

python manage.py makemigrations

python manage.py migrate

1. **Run the Django Development Server:**

bash

Copy code

python manage.py runserver

**Step 7: Frontend (Angular) Setup with KYC Document Upload**

1. **Create Angular Project:**

If you haven’t installed Angular CLI:

bash

Copy code

npm install -g @angular/cli

Then create a new Angular project:

bash

Copy code

ng new stock-simulation-frontend

cd stock-simulation-frontend

1. **Create Service for User Authentication (With KYC Upload):**

In src/app/services/auth.service.ts, create a service to handle user registration and KYC document upload:

typescript

Copy code

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class AuthService {

private apiUrl = 'http://localhost:8000/api/users/';

constructor(private http: HttpClient) { }

register(user: FormData): Observable<any> {

return this.http.post(this.apiUrl + 'register/', user);

}

login(user: any): Observable<any> {

return this.http.post(this.apiUrl + 'login/', user);

}

}

1. **Create Registration Form with File Upload:**

Create a component for user registration:

bash

Copy code

ng generate component Auth

Update src/app/components/auth/auth.component.html with the registration form:

html

Copy code

<h2>Register</h2>

<form (submit)="onRegister()" enctype="multipart/form-data">

<label>Username:</label>

<input type="text" [(ngModel)]="user.username" name="username" required>

<label>Email:</label>

<input type="email" [(ngModel)]="user.email" name="email" required>

<label>Password:</label>

<input type="password" [(ngModel)]="user.password" name="password" required>

<label>Role:</label>

<select [(ngModel)]="user.role" name="role">

<option value="trader">Trader</option>

<option value="regulator">Regulator</option>

<option value="company\_admin">Company Admin</option>

</select>

<label>KYC Document:</label>

<input type="file" (change)="onFileSelected($event)" name="kyc\_document">

<button type="submit">Register</button>

</form>

1. **Update the Auth Component for File Upload:**

In src/app/components/auth/auth.component.ts, handle the file upload:

typescript

Copy code

import { Component } from '@angular/core';

import { AuthService } from '../../services/auth.service';

@Component({

selector: 'app-auth',

templateUrl: './auth.component.html',

styleUrls: ['./auth.component.css']

})

export class AuthComponent {

user = { username: '', email: '', password: '', role: 'trader' };

selectedFile: File = null;

constructor(private authService: AuthService) { }

onFileSelected(event: any) {

this.selectedFile = event.target.files[0];

}

onRegister() {

const formData = new FormData();

formData.append('username', this.user.username);

formData.append('email', this.user.email);

formData.append('password', this.user.password);

formData.append('role', this.user.role);

if (this.selectedFile) {

formData.append('kyc\_document', this.selectedFile);

}

this.authService.register(formData).subscribe(

response => console.log('User registered successfully', response),

error => console.error('Error registering user', error)

);

}

}

**Step 8: Run Angular Development Server**

1. **Run the Angular App:**

bash

Copy code

ng serve

1. **Test Registration with KYC Document Upload:**

Open the Angular frontend, register a user, and upload a KYC document to ensure everything is functioning correctly.