**NAME : ABIRAMI R**

**REG NO : 22MCA0023**

**ROLE : BACKEND**

**Main code:**

# models.py

from django.db import models

from django.contrib.auth.models import User

class Alert(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

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

target\_price = models.DecimalField(max\_digits=20, decimal\_places=2)

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

triggered = models.BooleanField(default=False)

# serializers.py

from rest\_framework import serializers

from .models import Alert

class AlertSerializer(serializers.ModelSerializer):

class Meta:

model = Alert

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

# views.py

from rest\_framework import generics, permissions

from rest\_framework.response import Response

from .models import Alert

from .serializers import AlertSerializer

class AlertCreateAPIView(generics.CreateAPIView):

queryset = Alert.objects.all()

serializer\_class = AlertSerializer

permission\_classes = [permissions.IsAuthenticated]

def perform\_create(self, serializer):

serializer.save(user=self.request.user)

class AlertDeleteAPIView(generics.DestroyAPIView):

queryset = Alert.objects.all()

serializer\_class = AlertSerializer

permission\_classes = [permissions.IsAuthenticated]

class AlertListAPIView(generics.ListAPIView):

serializer\_class = AlertSerializer

permission\_classes = [permissions.IsAuthenticated]

def get\_queryset(self):

return Alert.objects.filter(user=self.request.user)

# urls.py

from django.urls import path

from .views import AlertCreateAPIView, AlertDeleteAPIView, AlertListAPIView

urlpatterns = [

path('alerts/create/', AlertCreateAPIView.as\_view(), name='alert-create'),

path('alerts/delete/<int:pk>/', AlertDeleteAPIView.as\_view(), name='alert-delete'),

path('alerts/', AlertListAPIView.as\_view(), name='alert-list'),

]

# email\_service.py

import smtplib

from email.mime.text import MIMEText

def send\_email(alert):

# Your email sending logic here

pass

# binance\_websocket.py

# Integration with Binance WebSocket for real-time updates

**Dockerfile:**

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

### **docker-compose.yml**

version: '3'

services:

web:

build: .

ports:

- "8000:8000"

depends\_on:

- db

- rabbitmq

db:

image: postgres:latest

environment:

POSTGRES\_DB: mydatabase

POSTGRES\_USER: myuser

POSTGRES\_PASSWORD: mypassword

rabbitmq:

image: rabbitmq:latest

ports:

- "5672:5672"

- "15672:15672"

### **requirements.txt**

Django==3.2.5

djangorestframework==3.12.4

psycopg2-binary==2.9.1

celery==5.1.2

redis==3.5.3

### **README.md**

# Price Alert Application

## Description

This is a price alert application built using Django, PostgreSQL, RabbitMQ, and Docker Compose. Users can create alerts for specific cryptocurrency prices and receive email notifications when the prices hit the target.

## Steps to Run

1. Install Docker and Docker Compose.

2. Clone this repository.

3. Navigate to the project directory.

4. Run `docker-compose up`.

## Endpoints

### Create Alert

- URL: `http://localhost:8000/alerts/create/`

- Method: POST

- Request Body:

```json

{

"coin": "BTC",

"target\_price": 33000

}

**Response Body (example):**

{

"id": 1,

"coin": "BTC",

"target\_price": 33000,

"created\_at": "2024-02-03T12:00:00Z",

"triggered": false

}

### **Fetch All Alerts**

[

{

"id": 1,

"coin": "BTC",

"target\_price": 33000,

"created\_at": "2024-02-03T12:00:00Z",

"triggered": false

},

{

"id": 2,

"coin": "ETH",

"target\_price": 2000,

"created\_at": "2024-02-03T13:00:00Z",

"triggered": true

}

]