**NAME : AAMEER ARSHATH A**

**REG NO : 22MCA0011**

**ROLE : BACKEND**

**Problem Statement**

Application Overview

Create a price alert application that triggers an email when the user’s target price is

Achieved.

Say, the current price of BTC is $28,000, a user sets an alert for BTC at a price of 33,000$.

The application should send an email to the user when the price of BTC reaches 33,000$.

Similarly, say, the current price of BTC is 35,000$, a user sets an alert for BTC at a price of

33,000$. The application should send an email when the price of BTC reaches 33,000$.

Things to do for the assignment

- Create a rest API endpoint for the user’s to create an alert `alerts/create/`

- Create a rest API endpoint for the user’s to delete an alert `alerts/delete/`

- Create a rest API endpoint to fetch all the alerts that the user has created.

- The response should also include the status of the alerts

(created/deleted/triggered/.. or any other status you feel needs to be included)

- Paginate the response.

- Include filter options based on the status of the alerts. Eg: if the user wanted

only the alerts that were triggered, then the endpoint should provide just that)

- Add user authentication to the endpoints. Use JWT tokens.

- There is no need to add tests.

- You can use Binance's WebSocket connection to get real-time price updates

- You can also use this endpoint to fetch the latest price of the cryptocurrency:

https://api.coingecko.com/api/v3/coins/markets?vs\_currency=USD&order=ma

rket\_cap\_desc&per\_page=100&page=1&sparkline=false but prefer using

Binance WebSocket

- When the price of the coin reaches the price specified by the users, send an email to the

user that set the alert at that price. (send mail using Gmail SMTP, SendGrid, etc). If this is

taking too much time, just print the output to the console.

- Add a caching layer for the “fetch all alerts” endpoints (use Redis/Memcache/etc)

Requirements

- You can use Python/ Go/ Ruby

- If you are applying for the Ruby role, please use Ruby.

- Use Postgres to store data (or any DB you feel that gets the job done)

- Use Rabbit MQ / Redis as a message broker for the task to send emails.

- Bundle everything inside a docker-compose file.

- Document your solution in the README.md file. Consider adding the following details

- Steps to run the project (eg: docker-compose up)

- Document the endpoints

- Document the solution for sending alerts

**SOLUTION:**

**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

}

]