Python Developer Technical Assignment

Stock Price Monitoring and Alert System

Overview

Create a real-time stock price monitoring and alerting system that demonstrates your ability to work with async programming, handle real-time data, implement caching, and follow object-oriented design principles.

Available Free APIs for Indian Stocks

1. **Yahoo Finance API (yfinance)**
   - No API key required
   - Completely free
   - Support for NSE stocks (add .NS suffix)
   - Install using: `pip install yfinance`
   - Example usage:

```python
import yfinance as yf

# For TCS from NSE
tcs = yf.Ticker("TCS.NS")
data = tcs.history(period="1d", interval="1m")

# For Reliance from NSE
reliance = yf.Ticker("RELIANCE.NS")
data = reliance.history(period="1d", interval="1m")
```

2. **NSE Data (nsetools)**
   - No API key required
   - Completely free
   - Install using: `pip install nsetools`
   - Example usage:

```python
from nsetools import Nse
nse = Nse()

# Get stock quote
quote = nse.get_quote('TCS')

# Get top gainers
```

```
top_gainers = nse.get_top_gainers()

# Get all stock codes
all_stocks = nse.get_stock_codes()
```

3. **NSE Python (nsepy)**
   - No API key required
   - Completely free
   - Historical data from NSE
   - Install using: `pip install nsepy`
   - Example usage:

```
from nsepy import get_history
from datetime import date

# Get historical data
data = get_history(symbol="SBIN",
                   start=date(2024,1,1),
                   end=date(2024,2,10))
```

Technical Requirements

Required Technologies

- Python 3.13+
- Redis (for caching)
- aiohttp (for async HTTP requests)
- pytest (for testing)
- Logging framework of your choice
- Type hints

Required Dependencies

```
aiohttp
redis
pytest
pytest-asyncio
yfinance
nsetools
nsepy
pydantic
```

System Components

1. Data Management System

```
from typing import List, Dict, Any
from pydantic import BaseModel
```

```python
from datetime import datetime

class StockData(BaseModel):
    symbol: str
    price: float
    volume: int
    timestamp: datetime
    high: float
    low: float
    open: float
    close: float

class DataFetcher:
    async def fetch_stock_data(self, symbol: str) -> StockData:
        """Fetch real-time stock data"""
        pass

    async def fetch_batch_data(self, symbols: List[str]) -> Dict[str, StockDa
ta]:
        """Fetch data for multiple symbols"""
        pass

class CacheManager:
    async def get_stock_data(self, symbol: str) -> StockData:
        """Get stock data from cache"""
        pass

    async def set_stock_data(self, symbol: str, data: StockData, ttl: int = 3
00):
        """Store stock data in cache"""
        pass
```

2. Alert System

```python
from enum import Enum
from typing import Callable

class AlertType(Enum):
    PRICE_THRESHOLD = "price_threshold"
    PERCENTAGE_CHANGE = "percentage_change"
    VOLUME_SPIKE = "volume_spike"

class AlertCondition(BaseModel):
    symbol: str
    alert_type: AlertType
    threshold: float
    comparison: str  # "above" or "below"

class AlertManager:
    async def add_alert(self, condition: AlertCondition):
```

```
        """Add new alert condition"""
        pass

    async def check_alerts(self, stock_data: StockData):
        """Check if any alerts should be triggered"""
        pass

    async def notify(self, alert: AlertCondition, stock_data: StockData):
        """Send notification for triggered alert"""
        pass
```

3. Monitoring System

```
class StockMonitor:
    def __init__(self, data_fetcher: DataFetcher, cache_manager: CacheManager
,
                 alert_manager: AlertManager):
        self.data_fetcher = data_fetcher
        self.cache_manager = cache_manager
        self.alert_manager = alert_manager
        self.is_running = False

    async def start_monitoring(self, symbols: List[str]):
        """Start monitoring specified symbols"""
        pass

    async def stop_monitoring(self):
        """Stop monitoring all symbols"""
        pass

    async def add_symbol(self, symbol: str):
        """Add new symbol to monitoring list"""
        pass

    async def remove_symbol(self, symbol: str):
        """Remove symbol from monitoring list"""
        pass
```

Required Features

1. **Data Fetching & Caching**
   - Implement real-time data fetching using any of the provided APIs
   - Cache data in Redis with appropriate TTL
   - Implement retry mechanism for API failures
   - Handle rate limiting appropriately
2. **Alert System**
   - Implement price threshold alerts
   - Implement percentage change alerts
   - Implement volume spike alerts

- Store alert history in Redis
- Implement console notifications (email optional)

3. **Monitoring System**
   - Monitor multiple symbols concurrently
   - Handle addition/removal of symbols dynamically
   - Implement graceful shutdown
   - Log all important events
4. **Error Handling**
   - Implement comprehensive error handling
   - Log all errors appropriately
   - Maintain system stability during API issues

Testing Requirements

1. **Unit Tests**

```python
# Example test structure
async def test_data_fetcher():
    fetcher = DataFetcher()
    data = await fetcher.fetch_stock_data("AAPL")
    assert data.symbol == "AAPL"
    assert data.price > 0

async def test_alert_manager():
    alert_mgr = AlertManager()
    condition = AlertCondition(
        symbol="AAPL",
        alert_type=AlertType.PRICE_THRESHOLD,
        threshold=150.0,
        comparison="above"
    )
    await alert_mgr.add_alert(condition)
    # Test alert triggering
```

2. **Integration Tests**

   - Test the entire system flow
   - Test error handling scenarios
   - Test concurrent operations

Example Usage

```python
async def main():
    # Initialize components
    data_fetcher = DataFetcher()
    cache_manager = CacheManager()
    alert_manager = AlertManager()

    monitor = StockMonitor(data_fetcher, cache_manager, alert_manager)
```

```python
    # Add alerts
    await alert_manager.add_alert(
        AlertCondition(
            symbol="AAPL",
            alert_type=AlertType.PRICE_THRESHOLD,
            threshold=150.0,
            comparison="above"
        )
    )

    # Start monitoring
    await monitor.start_monitoring(["TCS.NS", "RELIANCE.NS", "INFY.NS"])

    # Keep running for some time
    await asyncio.sleep(3600)

    # Stop monitoring
    await monitor.stop_monitoring()

if __name__ == "__main__":
    asyncio.run(main())
```

Evaluation Criteria

1. **Code Quality (20%)**
   - Clean, readable code
   - Proper use of type hints
   - Following PEP 8 guidelines
   - Proper documentation
2. **System Design (20%)**
   - Proper separation of concerns
   - Efficient use of OOP principles
   - Scalable architecture
   - Proper use of design patterns
3. **Error Handling (20%)**
   - Comprehensive error handling
   - Proper logging
   - System resilience
   - Graceful degradation
4. **Async Implementation (20%)**
   - Proper use of async/await
   - Efficient concurrent operations
   - Resource management
   - Performance considerations
5. **Testing (20%)**

- Comprehensive test coverage
- Well-structured tests
- Testing of edge cases
- Integration tests

Bonus Points (Additional 20%)

1. **WebSocket Implementation**
   - Implement WebSocket connection for real-time data
   - Handle WebSocket connection issues
   - Implement reconnection logic
2. **Performance Metrics**
   - Track and log system performance
   - Monitor memory usage
   - Monitor API call latency
   - Implement performance alerts
3. **Web Interface**
   - Simple Flask/FastAPI dashboard
   - Real-time price updates
   - Alert management interface
   - System status display

Submission Guidelines

1. **Code Repository**
   - Create a private GitHub repository
   - Add clear README with setup instructions
   - Include requirements.txt or poetry.lock
   - Include .env.example file
2. **Documentation**
   - System architecture document
   - API documentation
   - Setup guide
   - Testing guide
3. **Timeline**
   - 4 days (96 hours) from receiving the assignment
   - Submit GitHub repository URL
   - Include time spent on the project
   - Include a breakdown of time spent on different components (data management, alert system, testing, etc.)

Environment Setup

```
# Create virtual environment
python -m venv venv
```

```
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows

# Install dependencies
pip install -r requirements.txt

# Start Redis
docker run --name redis-cache -p 6379:6379 -d redis

# Run tests
pytest tests/

# Run application
python main.py
```

Hints

1. Use aiohttp for API calls instead of requests for better async performance
2. Implement exponential backoff for API retries
3. Use Redis pipeline for batch operations
4. Implement proper connection pooling
5. Use asyncio.gather for concurrent operations
6. Implement proper cleanup in **aexit** methods

Common Pitfalls to Avoid

1. Not handling API rate limits
2. Blocking operations in async code
3. Memory leaks in long-running processes
4. Not implementing proper error handling
5. Not cleaning up resources properly
6. Not handling edge cases in data processing

Remember to focus on code quality and system design rather than implementing every possible feature. A well-designed, stable system with fewer features is better than a poorly designed system with many features.