[개념질문**]**

1.

Q: 주식 또는 암호화폐 거래를 해본 경험이 있으신가요?

A:네 있습니다. 현재 암호화폐를 보유하고 있고 거래하고 있습니다.

2.

Q: 주식 & 옵션 & 선물에 대한 개념을 알고 계신가요?

A:

1) 주식: 회사의 소유권을 증명하는 문서. 즉, 회사의 주주가 되는 것이며 회사에 관한 의사결정을 할 수 있는 권리가 생기고 배당금을 수령할 수 있다. 주식거래소에서 쉽게 거래가 가능하다.
2) 옵션: 특정 시점, 특정 가격에 자산을 매매할 수 있는 권리이다. 매수할 권리를 콜옵션, 매도할 권리를 풋옵션이라고 한다. 옵션은 프리미엄을 지불해서 구매할 수 있다. 예를들어 A전자의 현재가가 1만원일 때, 3개월 후 주식을 1만 1천원에 매수할 수 있는 콜옵션이 있다고 가정하자. 프리미엄이 주당 500원이고 100주를 구매한다면 5만원을 지출하여 옵션을 구매할 수 있다. 이 때 옵션 행사 가격인 1만1천원을 행사가라고 하고 옵션이 만기되는 3개월후 특정 날짜를 만기일이라고 한다. 만약 만기일에 A전자의 가격이 1만 3천원이 돼서 옵션을 행사한다면 행사가인 1만 1천원으로 100주의 A전자 주식을 구매 할 수 있다. 구매한 주식은 주식 계좌로 들어오며 바로 팔 경우 주당 2천원의 이익을 볼 수 있다. 그런데 옵션은 권리이기 때문에 반드시 행사해야 하는 것은 아니다. 따라서 주식의 가격이 떨어져 옵션을 행사하지 않는다면 프리미엄으로 지출한 5만원만 손해를 보게된다.
3) 선물: 선물은 미래 특정 시점에 특정 가격으로 주식, 상품, 통화 등의 기초자산을 매매 할 것을 약속하는 거래이다. 예를들어 쌀이 현재 1KG 에 1만원일 때 1년 후 1KG당 2만원에 10KG을 매도할 선물계약을 체결한 경우 쌀의 가격이 어떻게 변하든지 10KG을 1KG당 2만원에 판매해야 한다. 선물은 옵션과 달리 계약이기 때문에 법적으로 반드시 거래가 이뤄져야 할 의무가 있다. 이 때 10KG을 계약 크기, 1년 후 날짜를 만기일이라고 한다. 선물거래는 미래 가격에 대한 약속이기 때문에 가격이 예상과 다르게 변할 경우 계약을 이행하지 않을 위험이 있다. 따라서 이를 방지하기 위해 보증금 개념인 초기 증거금을 매도자와 매수자 모두 예치 해야 한다. 계약을 유지하기 위한 최소 금액인 유지증거금이상 예치가 되어있어야 한다. 만약 가격 변동으로 유지증거금 이하로 떨어질 경우 마진콜이 발생하고 유지증거금을 유지 하기 위해 추가 금액을 예치해야 한다. 만기일에 정산이 되면 실제 자산이 매수자에게 전달되는 실물인수도가 발생한다.
   암호화폐 선물계약은 전통적인 선물계약과 다르게 만기일이 존재하지 않는 무기한 선물 계약이 존재한다. 만기일이 존재하지 않기 때문에 언제든지 정산이 가능하고 대부분의 경우 암호화폐 현물이 아닌 현금으로 정산한다. 선물을 매수(Long)한 경우 가격이 올랐을 때 정산하면 비율에 따른 현금 이익이 계좌로 들어오고 반대의 경우 계좌에서 현금이 빠져나간다. 무기한 선물은 펀딩비율이라는 중요한 개념이 있다. 펀딩 비율은 현물 가격과 선물 시장 가격의 차이로서 현물 자산의 가격과 무기한 선물시장의 가격을 일치시키기 위해 사용되는 메커니즘이다. 펀딩비율이 양수인 경우 매수(Long)을 보유한 사람들이 매도(Short)를 보유한 사람들에게 펀딩비용을 지불하고 음수인 경우엔 반대가 된다.

3.

Q: 만약 암호화폐 거래소를 사용하면서 떠오른 개선사항이 있다면 작성해주세요.

A: 없습니다.

4.

Q: 데이터 통신에는 API 통신과 Socket 통신이라는 두 가지 주요 유형이 있습니다. 이 두 통신 방식의 기본 개념과 그 차이점을 설명하고, 이러한 통신 기술을 활용한 개발 경험에 대해서도 간략하게 기술해 주세요.

A:

1) API 통신: 두 소프트웨어 시스템이 통신하는데 사용되는 일련의 프로토콜이다. 클라이언트-서버 구조를 사용하며 클라이언트가 서버에 요청해야만 값을 반환하는 단방향 통신을 사용한다. 대표적인 API 통신의 종류는 RestAPI와 GraphQL이 있다. Rest API는 클라이언트가 http 메소드를 사용해서 서버 엔드포인트(URL)에 CRUD작업을 요청하고 값을 반환받는다. GraphQL은 스키마를 사전에 정의해서 필요한 데이터만 정확하게 요청받을 수 있는 API통신의 한 형태이며 RestAPI와 달리 단일엔드포인트만 존재한다.
2) 소켓통신: 두 네트워크 노드간 직접적인 연결을 설정해서 데이터를 실시간으로 주고받는 방식이다. TCP, UDP 프로토콜을 주로 사용하며 두 노드간 지속적인 양방향 통신이 필요할 때 사용하는 통신 방법이다.
3) 차이점:
    a) 통신 방식:
    API 통신은 요청-응답 모델을 기반으로 하며, 클라이언트가 요청을 보내고 서버가 응답한다.
    Socket 통신은 실시간 양방향 통신이 가능하며, 지속적인 연결을 통해 데이터를 주고받는다
    b) 프로토콜:
    API 통신은 주로 HTTP/HTTPS 프로토콜을 사용한다.
    Socket 통신은 TCP 또는 UDP 프로토콜을 사용한다.
    c) 사용 사례:
    API 통신은 웹 서비스, 마이크로서비스, 클라우드 서비스 등에서 주로 사용된다.
    Socket 통신은 실시간 채팅, 온라인 게임, 실시간 데이터 스트리밍 등에 사용된다.
4) 개발 경험
    a) API통신
        - 닫닫닫의 OLOBO 프로젝트에서 Spring기반의 Rest API 서버 구축(API문서).
        - OLOBO서버에서 “Chat GPT API”사용하여 텍스트에서 감정 추출하는 기능 구현
    b) 소켓통신
        - 개인 프로젝트에서 java.net.Socket사용하여 채팅 서버 구현(github)

5.

Q: 암호화폐 거래소에는 어뷰징과 서버 트래픽을 관리하기 위해 Rate limit과 weight이라는 개념이 존재합니다. 아래의 개념을 참고하여, API와 Socket 통신 시 고려사항에 대해 작성해주세요. 개선사항을 제안해주셔도 좋습니다.

A:

1) 고려사항 및 개선사항
    a) API 통신: Rate Limit을 넘는 요청을 보내면 일정기간 통신을 할 수 없기 때문에 그것을 넘기지 않는 것이 중요하다. Weight은 Rate Limit과 달리 구체적인 수치가 없기 때문에 테스트를 통해 서버에서 사용할 분당 혹은 초당 최대 가중치를 산출해야 할 것 같다. 따라서 Rate Limit을 초과하지 않고, 적절한 Weigh의 관리를 위해, 클라이언트로부터 전송된 요청을 바로 처리하지 않고 큐(Queue)에 넣어 하나씩 처리하는 과정이 필요할 것 같다.

또한, 요청수를 체크하는 쓰레드를 배치하여 Rate Limit을 Weigh을 초과하지 않도록 모니터링 하는 시스템이 필요하다. 그리고 재시도 로직을 구현하여 실패한 요청만 따로 처리할 수 있는시스템이 필요하다. 이 때 서버 부담을 줄이기 위해 재시도 간격을 점차 늘리는 backoff전략을 사용해야 할 것 같다. 마지막으로 자주 요청하는 데이터에 대해 캐싱을 하는 것도 고려해 볼 수 있는 전략인 것 같다.

b) 소켓통신: 소켓 연결을 지속적으로 유지하고, 끊김이 발생했을 경우 자동으로 재연결하는 로직을 구현한다. 또한, 소켓 통신을 통해 수신하는 데이터의 빈도와 양을 모니터링하고, 필요시 데이터를 필터링하여 처리한다.

6.
Q: 직접 적인 쿼리문을 이용한 데이터 매핑 메서드에 대한 문제점과 대처법
A:

1) 문제점 및 대처법
   a) SQL 인젝션: 직접 쿼리문을 사용할 경우 사용자 입력이 쿼리문에 그대로 삽입되어 악의적인 사용자가 SQL인젝션 공격을 할 수 있다. 이를 방지하기 위해선 조건문 데이터를 직접 매핑하지 않으면 된다. 기본적으로 prepared statements를 사용해서 쿼리와 매개변수를 분리 및 이스케이프 처리하여 대처할 수 있다. 또한 내부적으로 prepared statements를 사용하는 sql mapper를 활용할 수도 있고 쿼리문 자체를 직접 작성하지 않기위해 쿼리 빌더를 이용하거나 ORM을 활용할 수도 있다.
   b) 코드 가독성 및 유지보수성 저하: 직접 쿼리문을 작성하면 코드가 복잡해지고 가독성이 떨어진다. 또한 SQL 구문이 코드에 직접 삽입되어 있기 때문에 쿼리를 변경하거나 수정하기가 어려워진다. 따라서 쿼리를 직접 작성하지 않는 쿼리빌더 혹은 ORM을 사용하여 이를 해결할 수 있다. 또한 복잡한 로직이 필요하다면 저장 프로시저를 사용하여 db가 쿼리를 관리하도록 할수도 있다.

7.
Q: Python 에서 멀티 쓰레딩을 구현하는 과정을 설명해주세요.
A: 파이썬 쓰레드는 threading.Thread클래스를 이용해서 구현할 수 있다. 쓰레드 객체를 직접 만들거나 threading.Thread를 상속해서 새로운 객체를 만드는 두가지 방식이 가능하다.

1) 쓰레드 객체를 직접 생성

```
import threading

# 스레드에서 실행할 함수 정의
def print_numbers():
    for i in range(1, 6):
        print(f"Number: {i}")

# 스레드 객체 생성
thread = threading.Thread(target=print_numbers)

# 스레드 시작
thread.start()

# 스레드 종료 대기
```

```
thread.join()
```

2) threading.Thread 상속받아 생성

```
import threading

class PrintNumbersThread(threading.Thread):
    def run(self):
        for i in range(1, 6):
            print(f"Number: {i}")

# 스레드 객체 생성
thread = PrintNumbersThread()

# 스레드 시작
thread.start()

# 스레드 종료 대기
thread.join()
```

멀티쓰레드는 프로세스의 메모리를 공유하므로 동기화 처리를 해줘야 한다.
파이썬은 threading.Lock 클래스를 사용하여 임계 영역을 보호할 수 있다.

```
import threading

# 공유 자원
counter = 0
lock = threading.Lock()

def increment_counter():
    global counter
    with lock:  # 임계 영역 시작
        temp = counter
        temp += 1
        counter = temp  # 임계 영역 종료

threads = []
for _ in range(10):
    thread = threading.Thread(target=increment_counter)
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()
```

여러개의 쓰레드를 직접 생성 및 관리하기보단 쓰레드풀을 사용하는 것이 자원 사용 및 구현 측면에서 효과적이다. 따라서 쓰레드풀을 관리하고 작업을 수행하는 concurrent.futures.ThreadPoolExecutor를 이용해서 멀티쓰레딩을 구현한다.

```
import concurrent.futures
```

```
import requests

# url에 http요청을 하는 함수. url과 status code를 반환
def fetch_url(url):
    response = requests.get(url)
    return url, response.status_code

# 요청할 url들
urls = [
    'https://www.example.com',
    'https://www.python.org',
    'https://www.github.com'
]

##### 쓰레드풀 생성
# 최대 3개의 쓰레드를 사용
with concurrent.futures.ThreadPoolExecutor(max_workers=3) as executor:
    ##### 작업 제출
    # fetch_url 함수를 executor가 관리하는 작업큐에 넣고, excutor가
    # 실행중이지 않은 쓰레드에 작업을 할당하여 수행한다. 작업 결과로
Future객체를
    # 반환 후 futures 리스트에 저장
    futures = {executor.submit(fetch_url, url): url for url in urls}

    ##### 완료된 작업 처리
    # 작업이 완료된 Future객체를 반환받아 url과 status code를 출력
    for future in concurrent.futures.as_completed(futures):
        url = futures[future]
        try:
            url, status = future.result()
            print(f'{url} returned status {status}')
        except Exception as exc:
            print(f'{url} generated an exception: {exc}')
```