# 🚩 이벤트 루프 아키텍처 충돌 이슈 보고서

## 문제 개요와 영향 (Issue Overview)

업비트 자동매매 시스템 전반에서 **이벤트 루프 충돌로 인한 기능 마비** 현상이 발생했습니다. 단순한 버그 수준을 넘어 **비동기 아키텍처의 근본적 결함**으로 판명되었습니다[[1]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L10-L16). 주요 증상은 다음과 같습니다:

* **다중 이벤트 루프 충돌** – PyQt6 GUI(QAsync 기반)와 개별 생성 asyncio.new\_event\_loop() 루프 간의 충돌로 예외 발생[[2]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L12-L15). 예를 들어, UI 컴포넌트가 별도 이벤트 루프를 만들고 aiohttp 세션 등을 호출하면서 충돌이 일어났습니다. 공용 리소스(HTTP 세션, asyncio.Lock 등)가 **다른 이벤트 루프에 바인딩**되어 동작 불능 상태가 되었습니다[[2]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L12-L15)[[3]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L84-L89).
* **전체 Infrastructure 계층 마비** – 이벤트 루프 충돌로 인해 **인프라 계층의 모든 API 호출 실패** 오류가 발생했습니다. 실제 오류 메시지는 *“<asyncio.locks.Lock object> is bound to a different event loop”* 형태로 나타났습니다[[4]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L26-L29). 이는 인프라 컴포넌트(UpbitPublicClient의 세션/락 등)가 **잘못된 루프에 묶여 호출되었음을 의미**합니다.
* **이슈 발생 시나리오** – 초기에는 호가창(Orderbook) 기능이 QAsync 기반 단일 루프에서 정상 작동합니다. 그러나 코인리스트 위젯을 열면서 별도 이벤트 루프를 생성하면 즉시 충돌이 발생합니다[[5]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L19-L27). 그 결과 **모든 Infrastructure Layer 호출이 실패**하고, 이후의 API 통신이 연쇄적으로 마비됩니다[[6]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L22-L30). 아래는 충돌 발생 과정의 요약입니다[[7]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L19-L28):
* **정상 작동 시작:** 호가창 서비스는 메인 QAsync 이벤트 루프에서 정상 실행됨 (DDD 규칙 준수).
* **문제 트리거:** 코인리스트 위젯이 별도로 asyncio.new\_event\_loop()를 생성하고 set\_event\_loop()로 교체하여 비동기 작업 수행 시도[[5]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L19-L27).
* **즉시 충돌:** Infrastructure 계층에서 공유하던 리소스들이 새 루프에 존재하지 않아 **모든 API 호출 실패**[[6]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L22-L30). 예: Lock 오브젝트가 다른 루프에 묶여있다는 오류 발생[[4]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L26-L29).
* **전역 기능 마비:** 주문 조회, 체결 조회 등 **전체 API 통신이 중단**되며 다른 컴포넌트도 동일 패턴 사용 시 동일한 문제가 예상됨[[8]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L30-L34).
* **영향 범위** – 호가창 기능은 QAsync 단일 루프 환경 덕에 영향을 받지 않았으나[[8]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L30-L34), 코인리스트 등 격리 루프를 사용하는 UI 기능은 **전면 마비**되었습니다. 이 패턴이 반복될 경우 향후 다른 모든 비동기 기능에도 충돌이 전파되어 시스템 안정성이 치명적으로 훼손될 수 있습니다[[9]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L32-L34).

## 근본 원인 분석 (Root Causes)

이슈의 핵심 원인은 **아키텍처적 설계 불일치**에 있습니다. 구체적으로, 이벤트 기반 UI 시스템과 인프라 계층 간에 철학이 충돌하고 개발 패턴의 일관성이 무너진 점이 지적됩니다:

* **설계 철학 충돌** – 프론트엔드 UI는 Qt 시그널/슬롯과 asyncio를 접목한 **이벤트 기반 설계**를 추구하지만, 백엔드 Infrastructure Layer는 **전역 공유 리소스**를 전제로 동작합니다[[10]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L72-L75). 한 애플리케이션 내에 서로 다른 비동기 패러다임(QAsync vs asyncio 표준 루프)이 공존하면서 **DDD 계층 간 정합성**에 문제가 발생했습니다. 즉, 도메인/인프라 계층은 하나의 일관된 실행 컨텍스트를 가정하는데, UI 레이어에서 임의로 루프를 분리하면서 계층 간 계약이 깨진 것입니다[[10]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L72-L75)[[11]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L77-L80).
* **개발 패턴의 일관성 부족** – 시스템 컴포넌트마다 **서로 다른 비동기 패턴**을 채택한 것이 원인입니다. 호가창 기능은 QAsync를 통해 **단일 이벤트 루프**를 사용했지만[[11]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L77-L80), 코인리스트 위젯은 별도 스레드+루프 격리 패턴을 사용했습니다. 이처럼 통일되지 않은 접근으로 인해, 새로운 컴포넌트마다 어떤 패턴을 쓸지 혼란이 생기고, 잘못된 조합 시 치명적 충돌을 일으키게 되었습니다[[11]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L77-L80). 다시 말해 **동일 시스템 내에 다중 이벤트 루프가 존재**하게 된 구조적 결함입니다[[12]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L64-L68).
* **인프라 계층 자원 초기화 문제** – Infrastructure 레이어의 핵심 클래스(UpbitPublicClient 등)들이 **이벤트 루프 의존적인 자원**을 가지고 있음에도, 이를 다중 루프 환경에서 공유하려다 문제가 생겼습니다. 예를 들어 UpbitPublicClient 초기화 시 생성되는 aiohttp.ClientSession 세션 객체나 Rate Limiter의 Lock은 해당 시점의 이벤트 루프에 바인딩됩니다[[3]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L84-L89). 그런데 코인리스트처럼 다른 루프에서 해당 인스턴스를 사용하면 **세션이나 락이 존재하지 않는 루프에서 호출**되어 예외가 발생합니다. 이러한 구조적 한계 때문에, 애초에 여러 이벤트 루프를 섞어 사용하는 현재 설계는 유지보수성과 안정성 면에서 무리가 있습니다[[3]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L84-L89).

요약하면, **이벤트 루프가 단일하지 않고 분리**됨으로써 발생한 모든 문제입니다. **도메인 주도 설계(DDD)** 원칙 하에서는 Infrastructure 리소스는 가능한 한 공유되고 일관되게 관리되어야 하지만, 이벤트 기반 UI와의 부조화로 그 원칙이 깨진 상황입니다. 이로 인해 시스템 전체에 **아키텍처적 부채**가 누적되고 있었습니다.

## 통합 이벤트 루프 전략(A) 채택 근거 (Solution Strategy A)

위 문제를 근본적으로 해결하기 위해 **QAsync 기반 단일 이벤트 루프 아키텍처(전략 A)**採用이 결정되었습니다[[13]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L242-L250). **모든 비동기 작업을 하나의 메인 이벤트 루프로 통합**함으로써 얻는 이점은 다음과 같습니다:

* **구조적 일관성 확보** – 전 시스템이 단일 이벤트 루프에서 동작하면, UI 레이어와 Infrastructure Layer 사이에 **맥락 불일치가 제거**됩니다. PyQt6의 GUI 이벤트 루프와 asyncio 루프를 QAsync로 결합하여 하나로 만들면, 컴포넌트 간 협업이 자연스러워지고 DDD 계층 간 계약이 회복됩니다[[14]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L96-L99). 즉, **모든 컴포넌트가 동일한 루프(맥락)**에서 돌기 때문에 앞서 지적된 철학 충돌이 사라집니다. 또한 Multi-loop로 인한 복잡한 동기화나 데이터 중복이 없으므로 아키텍처가 단순해집니다.
* **기술적 안정성 향상** – 이벤트 루프 충돌을 원천 제거함으로써 Lock 바인딩 오류 같은 문제가 재발하지 않습니다. 하나의 루프에 모든 비동기 리소스가 묶이므로 aiohttp 세션, Rate Limiter 등의 **공유 자원이 안전하게 재사용**됩니다[[14]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L96-L99). 또한 과도한 스레드 생성이나 run\_until\_complete 호출이 없어져 CPU 컨텍스트 스위칭 부담이 줄고, 메모리 누수 가능성도 감소합니다. 실제 기대 지표로는 **이벤트 루프 충돌 발생률 0%**, **API 호출 성공률 99.9% 이상** 등이 설정되었습니다[[15]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L274-L278). 이를 통해 시스템 호출 실패나 지연이 거의 없도록 보장합니다.
* **운영 및 유지보수 용이** – 단일 이벤트 루프 아키텍처는 운영 관점에서도 단순성과 효율을 제공합니다. 여러 개 루프와 스레드를 관리할 필요가 없으므로 **리소스 사용이 최적화**되고, 모니터링이 일원화되어 문제 파악이 쉬워집니다. 예를 들어, 전략 B(루프별 인스턴스 격리)를 택할 경우 루프마다 별도 세션/RateLimiter를 가져야 하며 메모리 중복 사용이 불가피하지만[[16]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L124-L129), 전략 A에서는 이런 **자원 중복과 오버헤드가 발생하지 않습니다**. 운영자는 하나의 이벤트 루프만 주시하면 되어 **시스템 상태 파악과 디버깅이 단순**해집니다. 개발팀 관점에서도 표준 패턴(QAsync)만 따르면 되므로 신규 기능 개발 시 패턴 선택의 혼란이 사라지고 생산성이 향상됩니다[[13]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L242-L250).

以上の理由から, **전략 A인 QAsync 단일 루프 통합**이 구조적/기술적/운영적 측면 모두에서 최적임이 확인되었습니다. 다만 구현을 위해 **기존 격리 패턴 코드의 대폭 수정**이 필요하고 QAsync에 대한 팀 학습이 요구되지만[[17]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L101-L104), 장기적으로 얻는 안정성과 일관성 이득이 그 비용을 압도합니다.

## 마이그레이션 단계 및 성공 지표 (Migration Plan & Success Metrics)

이행은 위험 최소화를 위해 **여러 단계로 구분**하여 진행됩니다.

* **1단계 – 긴급 조치:** 가장 치명적인 코인리스트 기능의 격리 루프를 즉시 제거하여 기본 기능을 복구합니다[[18]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L222-L229). 임시로 QAsync 패턴으로 전환해서라도 사용자에게 보이는 장애를 해소하는 것이 우선입니다.
* **2단계 – 단기 개선:** 아키텍처 방향을 **전사적으로 전략 A로 통일**하고 코딩 가이드라인을 수립합니다[[19]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L228-L236). 이 단계에서 Infrastructure와 이벤트 기반 시스템의 인터페이스를 재정비하고, Domain 이벤트 처리 방식 등 혼합된 패턴을 정리합니다. 코드 리뷰 체크리스트를 마련하여 이후 리팩토링에 일관성을 부여합니다.
* **3단계 – 전면 리팩토링:** 전체 시스템을 통합 이벤트 루프로 마이그레이션합니다[[20]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L234-L238). 모든 컴포넌트의 비동기 호출을 재점검하여 QAsync 기반으로 수정하고, Infrastructure 레이어도 필요시 리소스 초기화 방식을 보완합니다. 단계별로 컴포넌트를 수정하고 회귀 테스트를 거쳐 안정성을 확보합니다. 이 과정에서 점진적으로 배포하여 위험을 관리합니다.
* **4단계 – 안정화 및 평가:** 새로운 아키텍처로 **종합 기능 테스트**를 실시하고 모니터링 시스템을 통해 성능 지표를 추적합니다[[21]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L272-L278). 이벤트 루프 충돌이 재발하지 않는지, 응답 시간과 메모리 사용이 허용 범위 내인지 확인합니다. 최종적으로 불필요해진 과거 코드(예: 격리 패턴 legacy 모듈)를 제거하고 문서를 갱신합니다.

성공 여부는 사전에 정의된 **성공 지표(Success Metrics)**로 평가됩니다. 기술적 지표로는 *이벤트 루프 충돌 제로화*, *API 호출 성공률 99.9% 이상*, *메모리 사용량 20% 이내 증가*, *평균 응답 시간 10% 이내 증가* 등이 있으며[[15]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L274-L278), 개발 지표로는 *새 컴포넌트 개발 시 패턴 혼선 제거*, *아키텍처 관련 코드 리뷰 지적 감소* 등이 포함됩니다[[22]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L280-L283). 이러한 지표를 모두 충족할 때 비로소 마이그레이션이 성공적으로 완료되었다 판단할 것입니다.

# ⚙️ 실무 작업 가이드 (Refactoring Guide)

시스템을 전략 A에 맞게 리팩토링하기 위해, **주요 수정 작업을 몇 가지 패턴으로 구분**하여 설명합니다. 각 패턴별로 문제되는 코드 사례와 권장 수정 예시 코드를 제시하므로, 이 가이드를 따라 단계적으로 코드를 변경하면 됩니다. 마지막에는 **단계별로 실제 수정해야 할 파일 경로와 우선순위**를 정리합니다.

## 패턴 1: **격리 이벤트 루프 제거** (Isolated Loop Removal)

여러 이벤트 루프를 생성해서 병행 사용하는 패턴을 제거합니다. 이전에는 UI 작업을 백그라운드에서 처리하기 위해 asyncio.new\_event\_loop()를 만들고 별도 스레드에서 run\_until\_complete로 실행하는 방식이 사용되었습니다. 이 접근은 앞서 설명한 바와 같이 **다른 루프와 리소스 충돌**을 일으킵니다. 코인리스트 위젯 및 이벤트 시스템 초기화 코드에 이런 격리 루프 패턴이 발견됩니다.

**문제 코드 패턴:** 별도 이벤트 루프를 생성하여 동작시키는 예시입니다 (코인리스트 위젯 등에서 사용)[[23]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/upbit_auto_trading/ui/desktop/screens/chart_view/widgets/coin_list_widget.py#L232-L240). 새로운 asyncio 이벤트 루프를 만들고 현재 스레드의 루프로 설정한 뒤, run\_until\_complete로 비동기 함수를 동기 실행하고 있습니다. 이 과정에서 메인 루프와 분리된 루프가 생겨나게 됩니다.

# ⚠️ 격리된 새로운 이벤트 루프 생성 (문제 사례)  
new\_loop = asyncio.new\_event\_loop()  
asyncio.set\_event\_loop(new\_loop)  
result = new\_loop.run\_until\_complete(some\_async\_function())  
new\_loop.close()  
```【25†L231-L239】  
  
위와 같은 코드 대신, \*\*메인 QAsync 루프를 활용하여 비동기 함수를 실행\*\*하도록 수정해야 합니다. PyQt6 + QAsync 환경에서는 `asyncio.create\_task`나 `@asyncSlot`를 사용하면 별도 스레드나 루프 없이도 비동기 처리를 백그라운드로 수행할 수 있습니다. UI 이벤트 (예: 버튼 클릭, 타이머 시그널) 발생 시 그 핸들러를 async 함수로 정의해두면, Qt 메인 스레드의 QEventLoop에서 해당 함수를 안전하게 실행해 줍니다.  
  
\*\*수정 예시 코드:\*\* 코인리스트 위젯의 데이터 로드 함수를 QAsync 통합 패턴으로 변경한 예입니다. `@qasync.asyncSlot()` 데코레이터를 사용하여 PyQt 시그널에서 직접 비동기 함수를 호출할 수 있게 합니다. 별도 스레드 생성이나 `new\_event\_loop()` 호출 없이, \*\*메인 이벤트 루프에서 태스크로 실행\*\*되므로 충돌이 없습니다【1†L107-L115】:  
  
```python  
from qasync import asyncSlot  
  
class CoinListWidget(QWidget):  
 # ...  
  
 @asyncSlot()  
 async def \_load\_market\_data\_async(self):  
 """메인 루프에서 코인 목록 데이터 로드 (QAsync)"""  
 coins = await self.\_coin\_service.get\_coins\_by\_market(self.\_current\_market, self.\_search\_filter)  
 self.\_coin\_data = coins or []  
 self.\_update\_coin\_list() # UI 업데이트 동기 호출 (메인 스레드 안전)  
```【1†L107-L115】  
  
위 코드에서처럼 `asyncSlot`을 사용하면 Qt의 시그널(`QTimer.timeout`, 버튼 클릭 등)에 이 함수를 직접 연결할 수 있고, QEventLoop가 알아서 `await` 처리를 합니다. 결과적으로 별도 이벤트 루프를 만들 필요 없이 \*\*하나의 루프에서 비동기 작업들이 순차적으로 수행\*\*됩니다.   
  
코인리스트 위젯의 경우, 기존에는 `QTimer.singleShot`으로 1초 후 `\_load\_real\_data`를 호출하면서 그 내부에서 새로운 루프를 돌렸지만, 수정 후에는 `QTimer.singleShot`으로 `\_load\_market\_data\_async`를 호출하거나, 애초에 위젯 생성 직후 `await` 방식으로 데이터를 불러올 수 있습니다. UI 스레드에서 비동기 함수를 직접 호출할 수 없던 과거와 달리, 이제 QAsync 통합으로 \*\*메인 UI 스레드 자체가 비동기 루프를 지원\*\*하므로 이러한 패턴 전환이 가능합니다.  
  
또 다른 사례로 로그 뷰어(EventDrivenLogViewerWidget)의 이벤트 버스 초기화가 있습니다. 기존 코드에서는 별도의 이벤트 루프를 만들어 `event\_bus.start()`를 호출했지만【38†L201-L209】, 수정 시에는 위와 동일하게 `@asyncSlot`을 활용하여 메인 루프에서 `await event\_bus.start()`를 호출하면 됩니다. 이처럼 \*\*모든 격리 이벤트 루프 생성 코드를 제거\*\*하고, \*\*메인 루프에서 async 함수로 수행\*\*하는 것으로 수정 방향을 정리합니다.  
  
## 패턴 2: \*\*동기/비동기 혼합 호출 제거\*\* (Eliminate Sync/Async Mixing)   
이벤트 루프를 다루는 코드에서 \*\*동기적 블로킹 호출과 비동기 호출을 혼용\*\*하는 패턴을 제거합니다. 예전 코드에는 현재 루프가 실행 중인지 체크해서, 실행 중이면 `create\_task`로 태스크를 만들고 아니면 `run\_until\_complete`로 동기 실행하는 로직이 존재했습니다【40†L19-L27】. 이러한 코드는 불필요할 뿐 아니라, 잘못 사용할 경우 데드락이나 GUI 프리징을 유발할 수 있습니다.  
  
\*\*문제 코드 패턴:\*\* 도메인 이벤트 발행 등에서 사용된 혼합 패턴입니다. 현재 이벤트 루프 실행 여부에 따라 분기 처리하고 있는데, 통합 아키텍처에서는 이런 분기가 의미 없습니다【40†L19-L27】:  
  
```python  
# ⚠️ 동기/비동기 혼용 패턴 (문제 사례)  
loop = asyncio.get\_event\_loop()  
if loop.is\_running():  
 loop.create\_task(self.\_async\_publish(event))  
else:  
 loop.run\_until\_complete(self.\_async\_publish(event))  
```【40†L19-L27】  
  
이제는 애플리케이션이 항상 \*\*메인 루프(QAsync)\*\* 위에서 돌기 때문에, 위와 같은 분기 로직이 필요 없습니다. 언제나 이벤트 루프가 실행 중인 상황에서 코드를 호출하게 설계해야 하며, 따라서 `run\_until\_complete`와 같은 \*\*동기 대기 호출을 전부 제거\*\*합니다.   
  
\*\*수정 예시 코드:\*\* 이벤트 루프가 이미 실행 중임을 전제로, 모든 비동기 호출은 태스크로 생성하거나 상위 async 함수에서 `await`하도록 변경합니다. 위 예시는 다음처럼 단순화됩니다:  
  
```python  
# ✅ 항상 실행 중인 이벤트 루프에 태스크로 스케줄링  
asyncio.get\_event\_loop().create\_task(self.\_async\_publish(event))

혹은 현재 컨텍스트가 async 함수라면 그냥 await self.\_async\_publish(event)로 호출하면 됩니다. 중요한 것은 **절대로 run\_until\_complete를 호출하지 않는 것**입니다. 이 함수는 이벤트 루프를 동기적으로 막아버리고 다른 작업을 수행하지 못하게 하므로, GUI 어플리케이션에서는 사용을 지양해야 합니다. QAsync 환경에서는 asyncio.run(...)이나 run\_until\_complete 호출 없이도 자연스럽게 await로 비동기 처리 흐름을 구성할 수 있습니다.

동일한 맥락에서, 기존에 thread.join() 등을 통해 비동기 결과를 동기로 받던 코드도 모두 제거합니다. UI가 결과를 필요로 하면 asyncio.Future나 콜백, 신호를 통해 비동기 함수 완료를 통지받는 구조로 바꾸고, 호출 측도 가능한 한 async 함수로 전환하여 await를 사용하도록 리팩토링합니다. 결과적으로 시스템 전체에서 **동기 함수가 비동기 함수를 직접 호출하지 않도록** 만들어야 합니다. 모든 비동기 호출 경로는 이벤트 루프를 통해 이루어지며, 호출부도 async 함수이거나 적어도 create\_task로 시작하도록 표준화합니다.

## 패턴 3: **루프 인식 리소스 초기화** (Loop-aware Resource Initialization)

Infrastructure 계층의 리소스 생성 패턴을 점검하여 **현재 이벤트 루프와 정합성 있게 초기화**되도록 수정합니다. 예를 들어 aiohttp ClientSession이나 asyncio.Lock 객체는 생성 시점에 asyncio.get\_event\_loop()를 암묵적으로 참조합니다. 따라서 잘못된 시점에 생성하면 엉뚱한 루프에 묶이게 됩니다.

**문제 코드 패턴:** 이전에는 어플리케이션 시작 시점(메인 함수 실행 직후)에 인프라 객체들을 생성하면서 해당 시점의 이벤트 루프에 종속되는 문제가 있었습니다. UpbitPublicClient 초기화 코드상에서는 세션이나 RateLimiter 락을 즉시 생성하지 않도록 수정되었지만, 여러 루프 환경에서는 애초에 **인스턴스 단위를 루프별로 분리**하는 설계가 필요했습니다[[3]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L84-L89).

전략 A 채택 이후로는 **이러한 글로벌 리소스가 하나의 루프에만 묶이므로 문제 발생 가능성이 낮아집니다**. 그럼에도 불구하고, 혹시 모를 상황(예: 별도 스레드 사용)이나 테스트 환경을 대비하여 **리소스 초기화 지점을 명확히 하는 패턴**을 도입할 수 있습니다.

**수정 예시 코드:** 루프-의존 리소스를 다루는 하나의 방법은 **루프별 싱글톤 패턴**입니다. 즉, 현재 실행 중인 이벤트 루프 기준으로 인스턴스를 생성/캐싱하여 잘못된 참조를 방지하는 것입니다[[24]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L132-L140). 아래 코드는 이벤트 루프별로 UpbitPublicClient 인스턴스를 관리하는 예시입니다:

```python class UpbitPublicClientManager: \_instances: Dict[asyncio.AbstractEventLoop, UpbitPublicClient] = {}

@classmethod  
def get\_client(cls) -> UpbitPublicClient:  
 loop = asyncio.get\_event\_loop()  
 if loop not in cls.\_instances:  
 cls.\_instances[loop] = UpbitPublicClient()  
 return cls.\_instances[loop]

```[[25]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L134-L140)

위 매니저를 사용하면, 현재 루프에 맞는 UpbitPublicClient를 얻어올 수 있고 잘못된 루프에 바인딩된 세션을 쓰는 일을 막을 수 있습니다. 다만 이 패턴은 전략 B(루프 격리형) 아키텍처에 더 가까운 방안이므로, **전략 A 환경에서는 가능하면 하나의 글로벌 인스턴스만 사용**하도록 하는 편이 단순합니다. 실제 권장사항은 다음과 같습니다:

* 애플리케이션 시작 시 **메인 이벤트 루프가 설정된 이후**에만 인프라 리소스를 초기화합니다. 예컨대 QApplication과 QEventLoop 설정 직후에 UpbitPublicClient를 생성하거나, 아니면 첫 await 호출 시점까지 생성 시점을 늦춥니다.
* 이미 Infrastructure 객체들이 전역 싱글톤으로 관리되고 있다면 (예: get\_unified\_rate\_limiter()처럼), 해당 **싱글톤이 처음 생성되는 호출이 메인 루프 상에서 일어나도록** 유도합니다. 코인리스트 이슈에서는 별도 스레드에서 get\_unified\_rate\_limiter()를 호출해 RateLimiter가 잘못 초기화된 것이 문제였으므로, 이러한 함수들을 가능하면 앱 시작 시 한 번 미리 호출해 두거나 초기화 순서를 조정합니다.
* 혹시 향후 특정 컴포넌트에서 별도의 스레드+이벤트루프를 사용해야 할 경우(예: 극한의 I/O 병렬처리), 그 부분만 **전용 인프라 인스턴스**를 갖도록 설계합니다. 다만 이 또한 예외적 상황으로 간주하고, 기본 원칙은 **모든 공유 리소스는 하나의 이벤트 루프에서 운용**되도록 하는 것입니다.

결론적으로, **Infrastructure 계층의 객체 생성은 이벤트 루프와 시점에 민감하므로 최대한 lazy initialization** 기법을 활용합니다. 필요한 순간까지 객체 생성을 미루고, 생성 시에도 현재 asyncio.get\_event\_loop()가 가리키는 루프를 따르도록 해야 합니다. 이 원칙을 따르면 전략 A 하에서 리소스 바인딩 문제는 자연스럽게 해결되고, 만약 루프를 분리해야 하는 특별한 경우에도 안전하게 대응할 수 있습니다.

## 단계별 리팩토링 작업 목록 (Step-by-step Refactoring Plan)

마지막으로, 위 패턴들을 적용하기 위한 **실제 수정 작업들을 단계별로 정리**합니다. 우선순위가 높은 1단계부터 순차적으로 진행하며, 각 단계 내에서 파일 단위로 작업합니다. 괄호 안에 주요 수정 내용을 함께 표기합니다.

1. **1단계 – 긴급 수정 (격리 루프 제거 및 기본 기능 복구)**
2. upbit\_auto\_trading/ui/desktop/screens/chart\_view/widgets/coin\_list\_widget.py – **코인리스트 위젯 비동기 패턴 변경**. \_load\_real\_data에서 사용된 asyncio.new\_event\_loop() + 스레드 패턴을 제거합니다. 대신 @asyncSlot을 활용한 \_load\_market\_data\_async(가칭) 메서드를 도입하고, UI 초기화 시 해당 메서드를 호출하거나 타이머 시그널에 연결하여 **메인 루프에서 코인 데이터를 로드**하도록 수정합니다. 이 작업으로 코인리스트 기능 마비를 우선 해소합니다. (✅ **예상 난이도:** 높음, **우선순위:** 매우 높음)
3. upbit\_auto\_trading/ui/widgets/logging/event\_driven\_log\_viewer\_widget.py – **이벤트 로그 뷰어의 이벤트 시스템 초기화 방식 수정**. \_async\_setup\_event\_system에서 사용된 asyncio.new\_event\_loop() 및 run\_until\_complete 호출을 제거합니다. EventSystemInitializer.create\_simple\_event\_system() 호출부를 메인 루프 컨텍스트에서 실행하도록 리팩토링합니다. 구체적으로, 해당 부분을 별도 스레드가 아닌 **메인 쓰레드의 비동기 함수**로 전환하고, await event\_bus.start()로 호출되도록 변경합니다. 이로써 로그 뷰어 초기화 시 발생하는 잠재적 루프 충돌을 제거합니다. (✅ **난이도:** 중간, **우선순위:** 높음)
4. **2단계 – 아키텍처 표준화 적용 (혼합 패턴 정리 및 인프라 연계)**
5. upbit\_auto\_trading/infrastructure/events/domain\_event\_publisher\_impl.py – **도메인 이벤트 Publish 로직 개선**. publish() 메서드 내의 loop.is\_running() 분기와 run\_until\_complete 사용을 제거합니다. 항상 메인 이벤트 루프에서 동작한다는 전제로, 이벤트 발생 시 asyncio.create\_task(self.\_async\_publish(event)) 방식으로 이벤트 버스에 전달하도록 단순화합니다. 필요시 DomainEventPublisher 자체를 비동기 인터페이스로 변경하고, 호출부에서도 await publisher.publish(event) 형태로 사용할 수 있도록 수정합니다. 이로써 Domain 계층 이벤트가 전략 A 환경에 맞게 처리됩니다. (✅ **난이도:** 낮음, **우선순위:** 중간)
6. upbit\_auto\_trading/application/chart\_viewer/coin\_list\_service.py – **코인리스트 서비스 초기화 검토**. CoinListService에서 \_rest\_client = UpbitPublicClient()를 즉시 호출하는 부분이 있는데, 이는 메인 이벤트 루프 설정 후에 실행되므로 문제는 없습니다[[26]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/upbit_auto_trading/application/chart_viewer/coin_list_service.py#L82-L90). 그러나 일관성을 위해 이 초기화 패턴을 Orderbook 서비스와 유사하게 lazy하게 바꾸는 것을 고려합니다. 예를 들어 initialize() 비동기 메서드 내에서 UpbitPublicClient를 생성하거나, 또는 이미 즉시 초기화했더라도 그것이 메인 루프에서 이뤄지도록 보장합니다. (✅ **난이도:** 낮음, **우선순위:** 중간)
7. **인프라 전역 객체 초기화 순서 조정** – get\_unified\_rate\_limiter() 등이 처음 불릴 때 메인 루프에 바인딩되도록 애플리케이션 시작 루틴을 점검합니다. 예를 들어 ApplicationContext.initialize() 과정이나 메인윈도우 초기화 과정에서 한 번 호출해두는 방안을 고려합니다. 별도 코드 파일이 아닌 설정상의 변경이므로 구체 파일 경로는 없지만, **초기화 시퀀스 문서화**를 통해 개발자들이 인지하도록 합니다. (🔧 **난이도:** 낮음, **우선순위:** 중간)
8. **3단계 – 전면 마이그레이션 (전체 시스템 통합 루프 적용)**
9. **모든 UI 비동기 호출부 검토 및 수정** – UI 레이어 전반에 걸쳐 asyncio.run(...), run\_until\_complete, 스레드+루프 패턴 사용 여부를 검색하고 제거합니다. 예를 들어 차트뷰 화면의 다른 위젯들, 설정 화면 등의 비동기 처리 로직을 점검합니다. **표준 패턴은 “Qt 시그널 -> @asyncSlot 메서드 -> await 서비스 호출”**이므로, 이에 맞지 않는 코드를 발견하면 수정합니다. (📂 관련 폴더: upbit\_auto\_trading/ui/desktop/screens/, upbit\_auto\_trading/ui/widgets/) 각 컴포넌트가 공통 기반 클래스를 사용한다면 거기에 asyncSlot 활용 메서드를 도입하는 것도 고려합니다. (✅ **난이도:** 중간, **우선순위:** 중간)
10. **백그라운드 작업 처리 표준화** – 과거에 QThread나 스레드를 사용하던 작업 (예: 대용량 데이터 로드, DB 조회 등)이 있다면 가능하면 QAsync로 편입시킵니다. 필요시 asyncio.to\_thread 등을 활용하여 메인 루프에서 스레드 작업을 처리할 수도 있습니다. 핵심은 **UI와 백엔드 작업 간 스레드 통신을 모두 await/async로 대체**하여 일관성을 유지하는 것입니다. (예: 과거 worker.moveToThread 패턴은 우선 유지 가능하나, 점차 도메인 계층 비동기로 대체)
11. **인프라 계층 리팩토링** – Infrastructure의 네트워크 호출부 (UpbitPublicClient의 get\_\* 메서드들, WebSocketClient 등)가 **이벤트 루프에 직접 접근하지 않고** 순수 비동기로 동작하는지 확인합니다. 대부분 코루틴으로 구현되어 있으므로 이 부분은 문제가 없지만, 만약 내부에서 asyncio.get\_event\_loop()를 통한 루프 제어나 별도 태스크 생성 패턴이 있다면 제거합니다. 또한 필요하면 Rate Limiter의 락 사용부에 현재 루프와의 정합성 체크를 추가해 디버깅 도움을 주도록 합니다 (예: 잘못된 루프에서 호출 시 경고 로그). (✅ **난이도:** 중간, **우선순위:** 보통)
12. **종합 테스트 및 검증** – 코드 수정 후 전체 시스템 통합 테스트를 수행합니다. UI 쓰레드가 멈추지 않고 모든 비동기 작업이 잘 돌아가는지 확인하고, 다중 기능을 동시에 실행해보며 **이벤트 루프 충돌이 재발하지 않는지** 모니터링합니다. 특히 코인리스트와 호가창을 동시에 띄워서 모두 정상 동작하면 1차 성공입니다. (테스트를 자동화할 수 있다면 추가로 Selenium 등으로 GUI 시나리오 테스트를 구축)
13. **4단계 – 마무리 정리 및 성공 여부 확인**
14. **레거시 코드 정리** – 통합 루프 아키텍처로 전환하면서 더 이상 사용하지 않게 된 코드들을 제거하거나 격리합니다. 예컨데, .../chart\_view/widgets/legacy/coin\_list\_widget\_legacy.py 및 coin\_list\_widget\_new.py, coin\_list\_widget\_problematic.py 등 이전 실험적 구현 파일들은 프로젝트에서 제외합니다. 동일하게, 사용하지 않는 쓰레드 기반 유틸리티나 deprecated된 함수 (asyncio.run 호출부 등)도 삭제합니다. 이로써 코드베이스를 깨끗이 하고 혼란을 방지합니다. (🔥 **난이도:** 낮음, **우선순위:** 보통)
15. **성능 모니터링 및 튜닝** – 배포 후 실제 운영 환경에서 모니터링을 강화합니다. 이벤트 루프 단일화로 인한 **응답 지연이나 CPU 과부하가 없는지** 확인하고, 필요하면 asyncio.Task 스케줄링 최적화나 백프레셔 도입 등을 검토합니다. 또한 메모리 사용 변화를 추적하여 증가폭이 20% 이내인지 검증합니다[[15]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L274-L278). 목표 지표에 못 미치는 부분이 발견되면 원인을 분석해 튜닝 작업을 진행합니다 (예: UI 갱신 빈도 조절, Rate Limiter 알고리즘 조정 등).
16. **팀 공유 및 문서화** – 모든 수정이 완료되면 개발 가이드 문서를 업데이트하여 **새로운 비동기 패턴의 모범 사례**를 명시합니다. 특히 이번에 적용한 QAsync 통합 패턴에 대해 예제 코드와 함께 설명을 남깁니다. 코드 리뷰 절차에도 해당 아키텍처 원칙을 반영하여, 향후 유사 문제가 재발하지 않도록 합니다. 마지막으로 성공 지표 체크리스트를 점검하여, 각 항목을 충족했음을 확인하고 프로젝트 리더와 이해관계자들에게 결과를 보고합니다.

以上の 단계に従えば, 이벤트 루ープ アーキテクチャ危機를 성공적으로 극복하고 시스템의 안정성과 일관성을 확보할 수 있을 것입니다. 각 단계의 작업을 완료할 때마다 테스트를 병행하고, 문제가 발견되면 즉시 피드백을 반영하면서 진행하십시오. **이 문서 하나만으로도 리팩토링 작업을 수행할 수 있도록** 상세히 기술하였으므로, 차근차근 체크해가며 구현하면 될 것입니다. 모든 수정이 완료된 후에는 업비트 자동매매 시스템이 이전보다 훨씬 안정적으로 비동기 작업을 처리하게 될 것입니다.[[15]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L274-L278)[[22]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L280-L283)

[[1]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L10-L16) [[2]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L12-L15) [[3]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L84-L89) [[4]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L26-L29) [[5]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L19-L27) [[6]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L22-L30) [[7]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L19-L28) [[8]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L30-L34) [[9]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L32-L34) [[10]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L72-L75) [[11]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L77-L80) [[12]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L64-L68) [[13]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L242-L250) [[14]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L96-L99) [[15]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L274-L278) [[16]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L124-L129) [[17]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L101-L104) [[18]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L222-L229) [[19]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L228-L236) [[20]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L234-L238) [[21]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L272-L278) [[22]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L280-L283) [[24]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L132-L140) [[25]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md#L134-L140) EVENT\_LOOP\_ARCHITECTURE\_CRISIS\_20250926.md

<https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/docs/big_issues/EVENT_LOOP_ARCHITECTURE_CRISIS_20250926.md>

[[23]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/upbit_auto_trading/ui/desktop/screens/chart_view/widgets/coin_list_widget.py#L232-L240) coin\_list\_widget.py

<https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/upbit_auto_trading/ui/desktop/screens/chart_view/widgets/coin_list_widget.py>

[[26]](https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/upbit_auto_trading/application/chart_viewer/coin_list_service.py#L82-L90) coin\_list\_service.py

<https://github.com/invisible0000/upbit-autotrader-vscode/blob/5a2693266a9e9fda18b4a73dfd6da1ea47aa9563/upbit_auto_trading/application/chart_viewer/coin_list_service.py>