**시행착오2: 백테스팅 시스템 기능 구현 및 개선 과정**

**작성일:** 2025년 5월 12일

**프로젝트명:** 웹 기반 전략 백테스팅 시스템

**1. 개요**

본 문서는 웹 기반 전략 백테스팅 시스템 개발 과정에서 발생했던 주요 시행착오와 그 해결 과정을 기록하는 것을 목적으로 한다. 특히 사용자 계정 관리, 포트폴리오 백테스팅, 전략 유형 정의, PDF 리포트 다운로드 기능 구현 및 이후 발생한 오류 수정, UI/UX 개선 과정에서의 논의를 중심으로 정리한다.

**2. 주요 기능 구현 및 개선 과정에서의 시행착오**

**2.1. 사용자 계정 관리 기능 구현**

* **초기 상태**: users 테이블 스키마는 존재했으나, 실제 회원가입, 로그인, 로그아웃 및 사용자별 데이터 관리 로직은 부재.
* **시행착오 및 해결**:
  + flask\_login 라이브러리 및 werkzeug.security를 활용하여 기본적인 인증 시스템 구축.
  + UserMixin 클래스를 사용한 사용자 모델 정의 및 user\_loader 구현.
  + 회원가입 시 비밀번호 해싱, 로그인 시 해시된 비밀번호 검증 로직 추가.
  + @login\_required 데코레이터를 통해 접근 제어 구현.
  + strategies 및 results 테이블에 user\_id 컬럼 추가 및 외래 키 설정. 기존 로직에서 current\_user.id를 활용하여 사용자별 데이터를 저장하고 조회하도록 SQL 쿼리 및 백엔드 로직 수정. (이 과정에서 누락 시 다른 사용자의 데이터에 접근하거나, 데이터가 올바르게 필터링되지 않는 문제 발생 가능성 인지)

**2.2. 포트폴리오 백테스팅 기능 구현**

* **요구사항**: 단일 종목뿐 아니라 여러 종목으로 구성된 포트폴리오에 대한 백테스팅 기능 필요.
* **시행착오 및 해결**:
  + **UI 설계**: 여러 티커와 각 티커별 초기 투자 비중을 입력받기 위한 UI 고민. 초기에는 쉼표로 구분된 문자열 입력을 고려했으나, 사용자 편의성을 위해 동적 필드 추가/삭제 방식의 개선 아이디어 논의.
  + **백엔드 로직 설계 (portfolio\_backtest\_logic)**:
    - 여러 티커의 데이터를 동시에 로드하고 공통 거래일을 기준으로 데이터 정렬 및 전처리.
    - 각 자산에 초기 자본을 비중대로 분배하고, **단일 전략을 모든 자산에 동일하게 적용**하는 방식으로 단순화하여 접근.
    - **핵심 문제점**: portfolio\_backtest\_logic 함수 내에서 각 개별 자산(asset\_data\_df)에 대해 지표 계산 및 매매 신호 생성 로직이 **초기에 누락**되었음. 이로 인해 백테스트 실행 시 거래가 전혀 발생하지 않고 모든 결과 지표가 0으로 나오는 현상 발생 ("결과가 이상하네" 문제).
    - **해결**: run\_backtest\_logic의 지표 계산(SMA, RSI, MACD 등 pandas\_ta 사용) 및 신호 생성 로직을 portfolio\_backtest\_logic 내의 각 자산 처리 루프에 명시적으로 구현하고, NaN 값 처리 등을 통해 데이터 정합성 확보.
  + **결과 집계 및 표시**: 포트폴리오 전체의 수익률, MDD, 자산 가치 변화 등을 계산하고, backtest\_result.html 템플릿에서 포트폴리오 결과임을 명시적으로 구분하여 표시하도록 수정.
  + **데이터 저장**: results 테이블의 parameters JSON 컬럼에 포트폴리오 여부 (is\_portfolio: true), 티커 목록 (tickers), 비중 목록 (weights) 등을 저장하도록 execute\_backtest 함수 수정.

**2.3. "전략 유형" 명시적 정의 및 관리 기능**

* **요구사항**: 전략을 보다 체계적으로 분류하고 관리하기 위해 '전략 유형' 필드 추가.
* **시행착오 및 해결**:
  + strategies 테이블에 strategy\_type 컬럼 추가.
  + add\_strategy.html 및 edit\_strategy.html에 전략 유형을 선택할 수 있는 드롭다운 UI 추가 (사전 정의된 목록 사용).
  + app.py의 add\_strategy, edit\_strategy, list\_strategies 함수에서 strategy\_type 값을 처리하고 DB에 저장 및 조회하도록 SQL 쿼리 및 로직 수정.

**2.4. PDF 리포트 다운로드 기능 구현**

* **요구사항**: 백테스트 결과를 PDF 파일로 다운로드하는 기능.
* **시행착오 및 해결**:
  + **라이브러리 선택 및 설치**: WeasyPrint 라이브러리 선택.
  + **외부 라이브러리 의존성 문제**: WeasyPrint 실행 시 OSError: cannot load library 'libgobject-2.0-0': error 0x7e 발생. 이는 WeasyPrint가 의존하는 GTK+ 관련 C 라이브러리가 Windows 환경에 제대로 설치되지 않았거나 경로 설정이 미흡하여 발생한 문제.
  + **해결**: MSYS2를 사용하여 GTK3 및 관련 의존성 라이브러리(Pango, Cairo, GDK-PixBuf 등)를 설치하고, 해당 bin 폴더(C:\msys64\mingw64\bin)를 시스템 환경 변수 PATH에 추가하여 해결.
  + **PDF 템플릿 및 차트 생성**: PDF 내용을 담을 HTML 템플릿(pdf\_report\_template.html) 작성. Matplotlib으로 생성한 차트를 Base64로 인코딩하여 HTML에 직접 삽입하는 방식으로 PDF에 차트 포함. matplotlib.use('Agg') 설정으로 백엔드 환경에서 Matplotlib 사용 가능하도록 조치.
  + 한글 폰트 문제 해결을 위해 CSS에서 font-family 지정 및 서버 환경에 폰트 설치 필요성 인지.

**2.5. KeyError 발생 및 해결 (상세 보기 및 Excel 다운로드)**

* **문제 현상**: 포트폴리오 백테스팅 기능 추가 후, "백테스트 결과 목록"에서 포트폴리오 결과의 "상세" 버튼을 누르거나, 해당 결과에 대해 "Excel로 다운" 버튼을 눌렀을 때 KeyError: 'ticker' 오류 발생.
* **원인 분석**:
  + 단일 종목 백테스트 결과의 parameters에는 ticker (단수형) 키가 존재.
  + 포트폴리오 백테스트 결과의 parameters에는 tickers (복수형) 키로 티커 목록이 저장되고, 단일 ticker 키는 부재.
  + view\_result 함수와 download\_excel\_report 함수가 params['ticker']와 같이 단일 티커를 가정하고 작성되어 있어, 포트폴리오 결과 처리 시 해당 키를 찾지 못해 오류 발생.
* **해결**:
  + view\_result 및 download\_excel\_report 함수 내에서 params.get("is\_portfolio", False)를 통해 포트폴리오 결과인지 단일 종목 결과인지 판별.
  + 결과 유형에 따라 params['ticker'] 또는 params['tickers']를 참조하도록 로직 분기.
  + 템플릿 및 Excel 보고서 내용 생성 시에도 이 구분을 반영하여 ticker\_display 와 같이 일관된 변수를 사용하거나 조건부 처리를 통해 올바른 티커 정보(단일 또는 목록)를 표시하도록 수정.
  + KeyError 발생 가능성이 있는 딕셔너리 접근 시 .get()을 사용하거나, 키 존재 여부를 확인하는 방어 코드 추가.

**3. 결론 및 향후 과제**

기능 구현 과정에서 다양한 시행착오를 겪었으며, 특히 새로운 기능(포트폴리오 백테스팅) 추가 시 기존 기능과의 데이터 구조 차이로 인한 오류가 주요 문제점으로 나타났다. 각 기능의 특성을 고려한 데이터 모델링과 예외 처리, 그리고 명확한 로직 분기의 중요성을 다시 한번 확인하였다.

향후 UI/UX 개선을 통해 사용자 편의성을 증진시키고, 코드 리팩토링을 통해 유지보수성을 높이는 작업이 필요하다. 또한, 포트폴리오 백테스팅 기능의 경우, 리밸런싱 전략, 다양한 자산 배분 방법 등 고급 기능을 추가하여 시스템의 활용도를 더욱 높일 수 있을 것이다.