1. Inledning & syfte

Detta projekt syftar till att bygga en ETL-baserad applikation för att hantera och analysera en aktieportfölj där användaren får 1 mkr i fiktiva pengar att placera i valfria aktier på Stockholmsbörsens huvudlista (small, mid och large cap).

Projektet har i nuläget avgränsats för att fungera som en proof of concept och kan i ett senare skede utvecklas vidare. En sammanfattning kring delar som kan vidareutvecklas finns i slutet av denna rapport.

Projektet kombinerar tre centrala delar:

- ETL-flöde (Extract, Transform, Load): hämtar aktiedata från yfinance och lagrar den i en SQLite-databas.
- Databas & tjänstelager: strukturerad lagring av trades och prisdata, samt funktioner för att beräkna GAV, portföljvärde och avkastning.
- Visualisering i Streamlit: en interaktiv webbapplikation där användaren kan logga in (via demo-konto), registrera köp och sälj, samt följa portföljens utveckling i grafer och nyckeltal.

Målet med arbetet är inte att bygga en fullständig produktionsapplikation, utan en proof-of-concept (POC) som visar hur ett portföljflöde kan sättas upp i Python. Begränsningar har medvetet gjorts, till exempel att endast ett testkonto finns, att körningen inte är schemalagd samt att applikationen körs lokalt.

2. Genomgång av viktiga moduler

ETL (src/etl.py)

Denna modul ansvarar för att hämta kursdata via *yfinance*, formatera den och spara i SQLitedatabasen. Resultatet hamnar i tabellen prices som innehåller datum (ts), ticker och stängningskurs (close).

Syftet är att säkerställa att systemet alltid har uppdaterad marknadsdata. ETL-flödet kan köras manuellt i POC-versionen, men i en verklig tillämpning skulle det schemaläggas dagligen.

Databastjänster (app/services/db.py)

Hanterar uppkopplingen mot SQLite och ser till att tabellerna finns (trades, watchlist, prices). Förutom själva anslutningen används även pragman (inställningar) för att förbättra prestanda och säkerhet, exempelvis foreign_keys och journal_mode=WAL. Även här skulle det i en verklig tillämpning istället läggas i en molnbaserad databas, exempelvis via *Turso* eller någon av de stora cloud-tjänsterna (Azure, AWS, Google Cloud). På så sätt blir den schemalagda körningen oberoende av en specifik personlig dator.

Trades (app/services/trades.py)

Innehåller logiken för att spara trades. Här finns:

- Validering av indata (användare, ticker, datum, kvantitet, pris).
- Skydd mot att sälja fler aktier än man äger.

- Funktioner för att lista trades ur databasen.

Detta är grunden för att portföljen ska återspegla verkliga affärer.

Portfölj (app/services/portfolio.py)

Ansvarar för att beräkna användarens portföljstatus:

- positions() räknar ut hur många aktier användaren har av varje ticker.
- running_avg_costs() beräknar genomsnittligt anskaffningsvärde (GAV).
- cash balance() beräknar likvida medel utifrån startkapital och trades.
- realized/unrealized PnL räknar ut vinst/förlust både realiserad och orealiserad.
- overview() slår ihop allt till en översiktlig tabell som dashboarden kan använda.

Universe (app/services/universe.py)

Laddar listan över aktier från data/omx_securities.csv (Stockholmsbörsens huvudlista). Möjliggör sökning på företagsnamn. Detta används i trades-sidan för att underlätta för användaren att hitta rätt aktie eftersom namnen i yahoo finance inte alltid matchar vad man i Sverige kallar vissa av bolagen i vardagsspråk.

Streamlit-applikationen (app/)

Själva användargränssnittet.

- streamlit_app.py startvy med inloggning (demo-konto).
- pages/1_Dashboard.py visar KPI:er, innehavstabell och graf mot OMXSPI. Använder både portfölj- och prisdata.
- pages/2_Trades.py låter användaren registrera köp/sälj, hämta senaste kurs och se tradehistorik.
- pages/3_Models.py placeholder för framtida portföljmodeller (t.ex. Al/ML-modeller).

Övriga filer

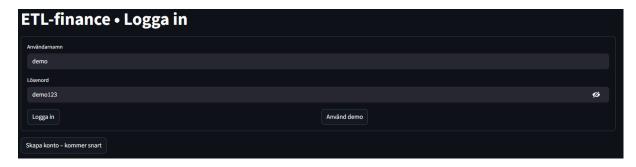
- config.py centrala konstanter (DB-sökväg, startkapital, demo-login).
- tests/test_etl.py verifierar att ETL fungerar genom att testa både dataformat och databasinladdning.
- data/omx_securities.csv univers av svenska aktier (namn, ticker, segment).

3. Exempel på flöde (use case)

För att illustrera hur applikationen fungerar följer här ett exempel med bilder där användaren köper aktier i Investor AB:

1. Login

Användaren loggar in via demo-kontot:





2. Registrera trade

På sidan *Trades* söker användaren fram *Investor B (INVE-B.ST)* ur universet och klickar på *Hämta senaste pris*:





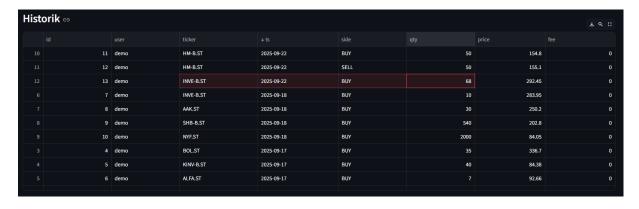


Kursen hämtas via yfinance och visas i fältet.

Användaren väljer BUY, anger kvantitet (68st i exemplet) och sparar affären:



- Affären valideras i trades.py (t.ex. att kvantitet > 0).
- Därefter sparas transaktionen i tabellen trades i databasen och i historiken.

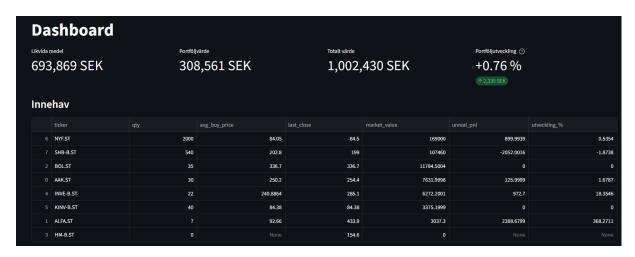


3. ETL-flöde

När ETL-skriptet körs uppdateras tabellen prices med de senaste stängningskurserna för Investor och övriga aktier i portföljen.

4. Dashboard

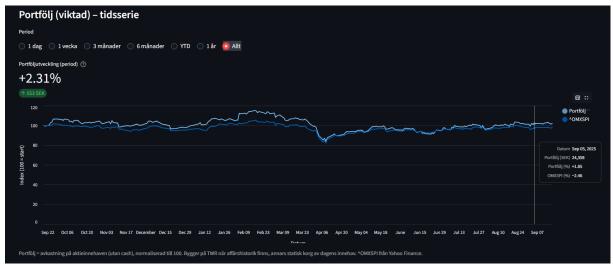
På Dashboard-sidan hämtas både trades och prisdata.



- portfolio.py räknar ut genomsnittlig anskaffningskostnad (GAV) och portföljens marknadsvärde.
- Orealiserad vinst/förlust beräknas baserat på skillnaden mellan senaste kurs och GAV.
- Likvida medel, portföljvärde och totalvärde.
- Innehavstabell där Investor nu finns med.

5. Resultat

Dashboarden visar även:



(Här har det tagits in historisk data för att kunna illustrera något längre historik).

- Graf som jämför portföljens utveckling med OMXSPI-index.
- Time-Weighted Return (TWR) används i grafen för att visa portföljens utveckling över tid.
- Även en vertikal crosshair med en tooltip när man hovrar över grafen visas.

Begränsningar och vidareutveckling

Det här projektet är utvecklat som en proof-of-concept (POC). Det innebär, som tidigare nämnt, att flera delar har förenklats eller medvetet avgränsats:

Begränsningar i nuvarande version

- Användarhantering det finns endast ett demokonto, och ingen möjlighet att skapa riktiga användare.
- Databas SQLite används lokalt, vilket är tillräckligt för en POC men inte för en applikation med flera samtidiga användare.
- Schemaläggning ETL-flödet körs manuellt, istället för automatiskt (t.ex. en gång per dag).
- Funktionalitet i modeller sidan *Models* är en placeholder och innehåller för närvarande inga modeller som användaren kan tillämpa på sin portfölj.
- Begränsad portföljlogik utdelningar, valutahantering och avancerade skatteregler är inte inkluderade.

Möjlig vidareutveckling

- Riktig användarhantering med registrering, krypterade lösenord och roller.
- Molnbaserad databas (ex. Turso, Azure SQL, AWS RDS) för att möjliggöra användning från flera datorer och samtidigt fler användare.
- Schemalagd ETL via cron job eller en molntjänst (t.ex. GitHub Actions eller Airflow).
- Utökad portföljlogik, t.ex. stöd för utdelningar, fonder, ETF:er och valutahandel (och även handel av utländska aktier som kräver att valutahandel finns eftersom man alltid börjar med en viss valuta).
- Portföljmodeller och AI som kan föreslå rebalansering eller simulera olika strategier. För detta kan det krävas att man lägger in historisk data i databasen. Det är dock viktigt att avgöra för hur många aktier och hur långt tillbaka i tiden som skulle kunna ge bra historik för olika modeller.
- Deployment till Streamlit Cloud eller liknande, så applikationen går att nå online.

- Reset-möjlighet för användaren om den vill starta om med "clean slate", enkel funktion att lägga in men viktig.

4. Slutsats

Projektet visar hur ett ETL-baserat system kan användas för att hantera och analysera en aktieportfölj. Genom en kombination av datainsamling från yfinance, lagring i SQLite och visualisering i Streamlit har en fungerande helhetslösning skapats.

Systemet visar att det är möjligt att bygga en modulär portföljtracker där data flödar från ETL-process till databastjänster och vidare till presentation i ett användargränssnitt. Även om projektet är avgränsat till ett proof-of-concept finns det en tydlig grund för vidareutveckling. Viktiga nästa steg vore att lägga till riktig användarhantering, hantering av utdelningar och fonder samt att köra ETL-flödet automatiskt i molnet.