**Custom Equal-Weighted Index Project Documentation**

**1. Project Overview**

**Objective:**

* Construct an equal-weighted custom index comprising the top 100 US stocks by market cap and update it daily.
* Data Ingestion: Retrieve daily stock data using yfinance.
* Data Storage: Persist data in a file-based SQLite database.
* Index Calculation: Compute an equal-weighted index by assigning each constituent an equal weight.
* Dashboard Visualization: Provide an interactive dashboard (using Streamlit) to display the index’s performance and composition.
* Export Capability: Export index performance and composition data to Excel and PDF formats.

**Key Design Principles:**

* **Equal Weighting:** Each stock contributes equally (e.g., each of 100 stocks gets a 1% weight).
* **Daily Rebalancing:** Recalculate and rebalance the index each trading day.
* **Idempotency:** Ensure that re-running the ingestion job does not duplicate data.
* **Robust Data Handling:** Account for missing data using watermark logic and proper error handling.

**2. System Architecture and Data Pipeline**

**Data Ingestion:**

* **Source:**
  + Use yfinance to download daily historical price data.
  + Optionally integrate Finnhub to fetch current fundamental data such as market cap.
* **Ticker Universe:**
  + Initially sourced from the NASDAQ Trader file (e.g., ftp://ftp.nasdaqtrader.com/SymbolDirectory/nasdaqtraded.txt) with filters applied to include only active stocks.

**Data Storage:**

* **Database:**
  + A file-based SQLite database (e.g., “data.db”) is used to persist data. The database file is created automatically if it doesn’t exist.
* **Tables:**
  + **stocks:** Stores ticker symbols and optionally names.
  + **daily\_data:** Stores daily data with columns for date, ticker, closing price, and market cap.
* **Idempotent Inserts:**
  + Uses INSERT OR IGNORE and INSERT OR REPLACE to prevent duplicate entries on reingestion.

**Index Calculation:**

* **Equal-Weighted Index Calculation:**
  + For each trading day, the top 100 stocks (by market cap) are selected.
  + The index value is calculated as the average of the closing prices of these stocks.
* **Daily Rebalancing:**
  + The index is recalculated daily to adjust for any changes in constituent stocks.
* **Date Handling:**
  + SQL queries use SQLite’s DATE() function to extract the date portion when stored timestamps include time.

**Dashboard Visualization:**

* **Streamlit Dashboard:**
  + Provides UI elements (date range selectors, forms) for user interaction.
  + Displays index composition and performance data.
  + Visualizations include a table for composition, a bar chart for top stocks, and a line chart for index performance.

**3. Detailed Module Descriptions**

**a. DataFetcher (data\_fetcher.py)**

**Purpose:**

* Fetches historical stock data using yfinance.

**Key Methods:**

* **init**: Initializes with a list of tickers, a CSV file, or a default online source.
* load\_all\_us\_tickers: Downloads the NASDAQ Trader file and filters to obtain active US stocks.
* get\_market\_cap\_finnhub: Fetches market cap from Finnhub for a given ticker.
* fetch\_data: Downloads daily data for each ticker for a specified date range.

**b. DatabaseManager (database\_manager.py)**

**Purpose:**

* Manages SQLite database operations.

**Key Features:**

* Creates the tables “stocks” and “daily\_data” using “CREATE TABLE IF NOT EXISTS”.
* Provides idempotent data insertion methods.

**c. CustomIndexCalculator (custom\_index\_calculator.py)**

**Purpose:**

* Calculates the equal-weighted custom index.

**Key Methods:**

* calculate\_index\_value(date): For a given date, queries the top 100 stocks by market cap and returns the average closing price.
* calculate\_index\_for\_date\_range(db\_manager, start\_date, end\_date): Iterates over trading days (using pd.bdate\_range) within a specified range to compute daily index values.

**d. Exports (exports.py)**

**Purpose:**

* Exports DataFrame data to Excel and PDF.

**Key Functions:**

* export\_to\_excel(df, filename): Saves a DataFrame as an Excel file.
* export\_to\_pdf(df, filename): Generates a PDF report from a DataFrame using ReportLab.

**e. Dashboard (dashboard.py)**

**Purpose:**

* Provides an interactive user interface for visualizing index composition and performance.

**UI Features:**

* Date range selection (with an optional submit button via a form).
* Displays a table of index composition over the selected date range.
* Visualizes the top 10 stocks by average market cap (bar chart) and overall index performance (line chart).

**4. Design Decisions and Rationale**

* **Equal-Weighted Index:**  
  Ensures each stock has an equal influence, avoiding the dominance of larger companies.
* **Daily Rebalancing:**  
  Reflects the daily changes in the market and adjusts the constituent list accordingly.
* **Persistent Database (SQLite):**  
  Provides a lightweight solution that requires no separate server.  
  Supports idempotent data ingestion, ensuring that re-running the job does not duplicate data.
* **yfinance Integration:**
  + yfinance is used for historical price data.
  + Finnhub adds value by providing fundamental data such as market cap (when available).
* **Interactive Dashboard with Streamlit:**  
  Offers a user-friendly interface for data visualization.  
  Allows users to easily adjust parameters (e.g., date ranges).
* **Robust Error Handling:**  
  Incorporates logging and error checks to ensure the system is resilient to data gaps and ingestion issues.

**5. Conclusion**

This solution delivers an end-to-end pipeline for constructing an equal-weighted custom index by:

* Ingesting data from yfinance (with optional Finnhub support).
* Persisting data in a file-based SQLite database.
* Calculating the index by assigning equal weights and rebalancing daily.
* Providing interactive visualization via a Streamlit dashboard.
* Enabling data export to Excel and PDF.

The design strikes a balance between simplicity and robustness, ensuring ease of maintenance, scalability, and accurate market representation.