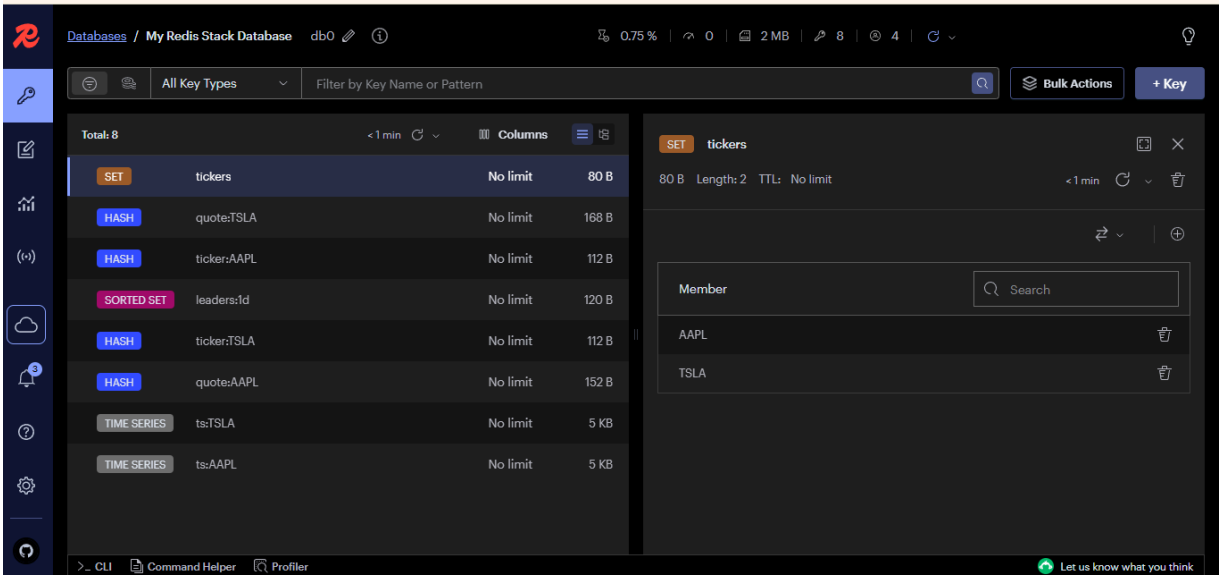
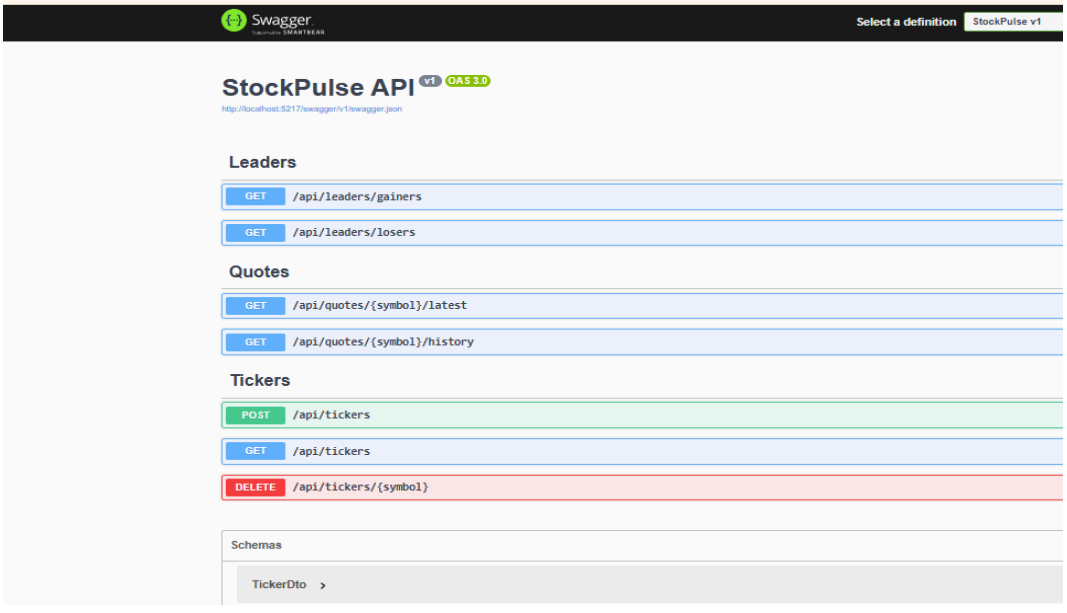
📈 StockPulse

Github: https://github.com/KhawajaAbdullah2000/StockPulse_Dot_Net_Redis/



1. Project Overview

StockPulse is a **.NET 8 Web API** project designed to simulate real-time stock price tracking using **Redis** as a high-performance in-memory database. The project demonstrates how financial applications can manage fast-moving data efficiently while maintaining scalability and responsiveness. This application uses **Redis** TimeSeries accessed using **Docker** for storing stock price history and sorted sets for leaderboards, providing a practical and modern backend solution.



2. Problem it Solves

Traditional relational databases struggle when handling **high-frequency financial data** such as stock price updates. Operations like calculating top gainers/losers or storing historical data can become slow and resource-intensive. StockPulse solves this by leveraging **Redis**, which provides:

- **High-speed data ingestion** for stock prices.
- **Efficient retrieval of time-series data** for historical trends.
- **Real-time leaderboards** to identify market winners and losers instantly.
  This makes it ideal for financial analytics, dashboards, or learning how real trading systems might manage data.

3. Technology Stack

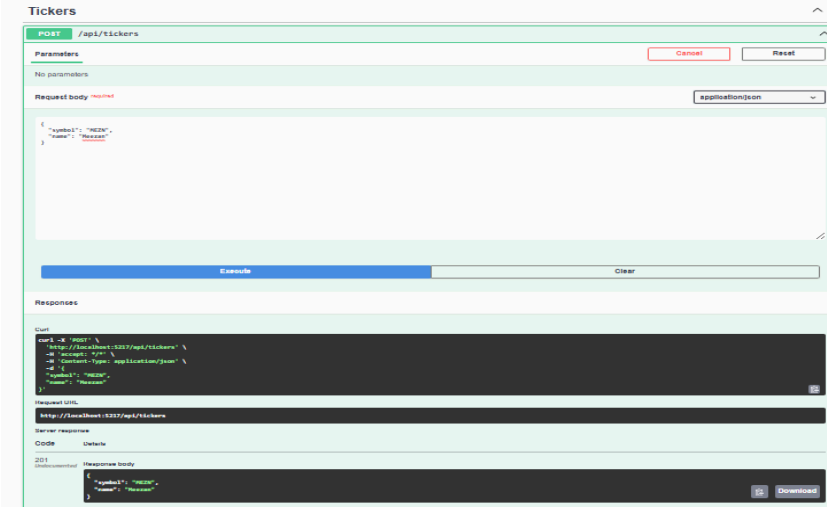
- **.NET 8 Web API** – Core backend framework.
- **C#** – Programming language.
- **Redis (via Docker)** – In-memory database with TimeSeries and Sorted Sets.
- **StackExchange.Redis** – Redis client for .NET.
- **NRedisStack** – Provides Redis modules support (TimeSeries, Bloom, Graph, etc.).
- **Docker** – Used to run Redis locally in a containerized environment.

4. Key Features

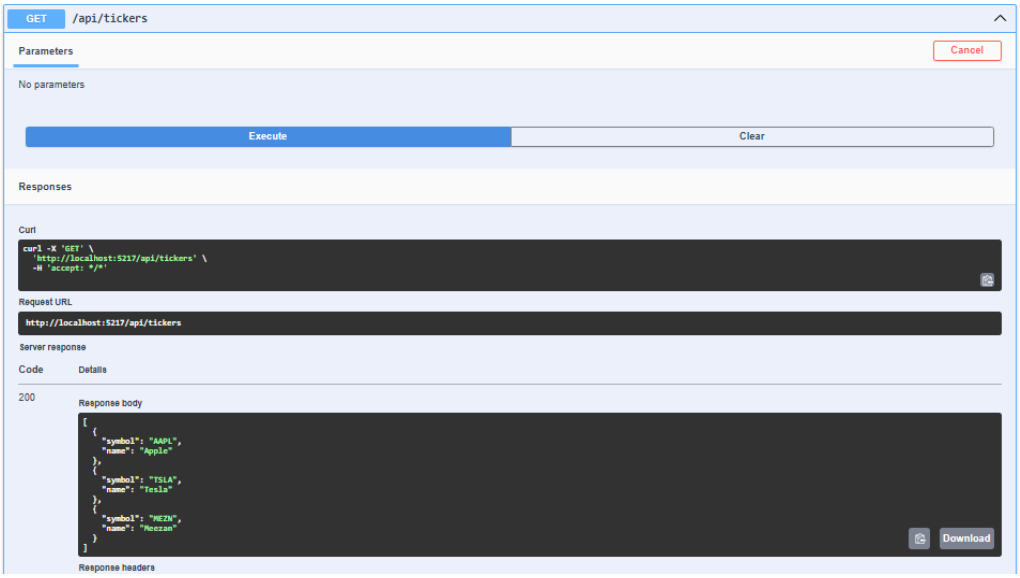
- 📈 **Add stock prices** in real time via API.
- 📜 **Retrieve stock price history** using Redis TimeSeries.
- 🏆 **Leaderboard APIs** to fetch top gainers and top losers.
- 🐳 **Dockerized Redis setup** for quick and portable development.
- 🔌 **RESTful endpoints** following clean architecture principles.

5. How it Works

- A user or service sends a stock update (e.g., symbol = `MEZN`, name= `Meezan`) to the API.



- The API stores this in **Redis TimeSeries**, maintaining a full price history.
- The API also updates a **Redis Sorted Set** to track percentage changes for leaderboard calculations
- Users can query:
  -Stock history (`/api/stocks/history/{symbol}`)
  -Top gainers (`/api/stocks/leaders?losers=false`)
  -Top losers (`/api/stocks/leaders?losers=true`)



- Redis, being in-memory, ensures **fast reads/writes** and supports real-time analytics.