Market Data Server Coding Challenge

Overview

The purpose of this coding challenge is to assess a candidate's ability to work with real financial market data, design a clean data pipeline, and expose that data via a simple but well-structured API.

The task mirrors the type of work carried out at ActrixFT: consuming public market data, normalising it into a usable analytics layer, and making it accessible to downstream applications.

Candidates are expected to spend **approximately 8 hours** in total – i.e. a "9am-5pm" type of project.

Objective

Build a simple **market data service** that:

- Ingests constant maturity government bond yields
- Normalises yields across maturities
- Exposes the data via a REST API
- Demonstrates the API working in a browser

A small visual application demonstrating the API is an optional stretch goal.

Time Expectation

- Core requirements: ~6 hours
- Stretch goal: ~2 hours

We value clarity, correctness, and good engineering judgement over visual polish.

Technology Stack

The choice of tools is flexible, but the following are preferred:

- **Language**: Python
- **Backend framework**: FastAPI, Flask, Express
- **Storage**: In-memory, SQLite, or Flat files
- **Frontend (optional)**: Streamlit, Dash, simple HTML/JavaScript, or React
- **Deployment**: Local execution only (no cloud deployment required)

If assumptions are required, candidates should document them clearly.

Part 1 – Data Ingestion (Core)

Data Sources

1. **US Treasuries**
   Source: Federal Reserve Economic Data (FRED)
   Use constant maturity Treasury yields (e.g. 1M, 3M, 6M, 1Y, 2Y, 5Y, 10Y, 30Y).

2. **UK Gilts**
   Source: Bank of England
   Use nominal par yields or zero-coupon yield curve data.

Requirements

- Fetch data programmatically (Source API or downloadable CSV)
- Store as time series with at least the following fields:
    - Date
    - Country (US, UK)
    - Instrument (Treasury, Gilt)
    - Maturity (expressed numerically in **years**)
    - Yield
- Storage can be Flat files or SQL DB
- Handle missing days gracefully (e.g. weekends, holidays)
- Store at least 2 years of historical data
- Do not hard-code data

Part 2 – Curve Normalisation (Core)

Objective

Expose yields on a continuous maturity grid from **1 day** out to **50 years**.

Requirements

- Interpolate yields across maturities
- Allow queries for any maturity within the supported range
- Choice of interpolation method is left to the candidate (e.g. linear, cubic, Nelson-Siegel)

The emphasis is on clean implementation and clear explanation of design choices.

Part 3 – REST API (Core)

Required Endpoints

1. Latest Yield

GET /latest

Query parameters: - country=US|UK - maturity=years (float, e.g. 2.5)

Example response:

```
{
  "date": "2025-01-15",
  "country": "US",
  "maturity": 2.5,
  "yield": 0.0423
}
```

2. Historical Time Series

GET /timeseries

Query parameters: - country=US|UK - maturity=years - start_date - end_date

Example response:

```
{
  "country": "UK",
  "maturity": 10,
  "data": [
    {"date": "2024-01-02", "yield": 0.041},
    {"date": "2024-01-03", "yield": 0.0408}
  ]
}
```

API Expectations

- Load data from store (Part 1) and interpolate
- Clear and consistent JSON responses
- Sensible defaults
- Meaningful error messages
- Fast response times

Part 4 – Demonstration (Core)

Candidates must demonstrate that:

- The server starts and runs cleanly
- Endpoints can be queried from one of:
    - A web browser
    - Or curl or equivalent

At minimum, provide example URLs in the README.

Candidates will be expected to demo and talk through/explain their final code.

Part 5 – Stretch Goal (Optional)

Build a small application demonstrating what the market data server can do.

Examples include: - Yield curve snapshot for US and UK - Time series chart for a selected maturity - Interactive maturity selection (e.g. slider from 1 day to 50 years)

Suggested tools include Streamlit or Dash.

Deliverables

Candidates should submit:

1. Source code (ideally a personal GitHub repository)
2. README file describing:
    - How to run the server
    - Data sources used
    - Design and implementation choices
    - What they would improve with more time
3. Demonstration to ActrixFT.

Additional Context

Assume that this service may later be consumed by pricing, risk, or analytics systems.