We are excited to learn that you are interested in building the next generation of financial technology here at Apollo Global Management. There is a growing need for technical talent at our firm. The following tasks illustrate practical problems that our technologists are required to solve on a day to day basis.

Your solution for these tasks should follow industry standards for production quality code. This includes concise code that is easy to read, functional and object-oriented programming paradigms with standardized file structures and naming conventions. Additionally, we look for smart use of external dependencies, you should not have to reinvent the wheel. We will look favorably upon solutions that employ practical unit tests and generics.

Your solution must be well documented enough to follow your logic and compile and run inside a virtual machine. Our programmers are polyglots, and our teams consist of experts in python and JVM (Java, Scala, Kotlin, Groovy) languages. Answers in these languages are strongly preferred.

**Task 1 - ETL of Semi-Structured Data**

Data from external vendors is often provided via a flat file format with limited customization options for receiving raw data. Given vendor specifications, we most load their data into internal systems to undergo more advanced analytics. The attached xyz\_emsx.csv is an example of trading data produced by internal users from an external vendor product. Opening the file you will notice there are three header rows with the row type being the left most column (ORDER, ROUTE, FILL). Each row type represents its own table type. Load these three tables into a lightweight development database such as SQLLITE or H2 or your own technology of choice. This dependency should be clearly documented, and you should be able to defend your architecture decisions. We must be able to run your solution without standing up additional infrastructure outside your packed code.

Notice different column have different data types. Your tables should be defined with the appropriate types. Number types should be precisely defined, and dates should be properly parsed.

This data should then be exposed via API to be shared across other services. Build a webserver that exposes this data via RESTful API. Well documented API endpoints with practical request contracts must return the following data objects defined on the next page. These JSON objects will ultimately be used to evaluate the accuracy of your solution.

To ensure we can validate your JSON, please document how to build and start your webserver, how it loads the data, and your endpoint request contracts. Functional example endpoints that are provided should work with common tools such as curl or postman.

1. Parent Order Object and Underlying fill data. Given: order\_id Return: top level order object with grouped fill objects.

Expected JSON:

{

“order\_id”: int .

“ticker”: string,

“side”: string,

“amount”: int,

“create\_date\_time”: ISO 8601 datetime with NYC timezone ie: (2022-06-05T22:22:00+00:00),

“security\_name”: string.

“filled\_amount”: int,

“fills”: [{

“fill\_price”: float,

“fill\_amount”: int,

“fill\_as\_of\_date\_time: ISO 8601 datetime string

}…]

}

2. Additional query options to search orders by ticker, and by date range (ISO date, YYYY-MM-DD). For the date range, we expect a list of JSON objects. Please follow the above JSON structure.

3. Returning Order summary statistics, given all the above defined search parameters (order\_id, ticker, and date range, return the following JSON for a given order.

{

“order\_id”: int,

“ticker”: string,

“average\_price”: float, (take the mean of the fill price)

“std\_price”: float, (take the standard deviation of the fill price)

“fill\_duration”: (int, total seconds for an order to be filled, return null for part-filled orders)

}

**Task 2 – Beta Calculation:**

Implement a function for beta calculation with respect to S&P 500 index (SPY US Equity) for the stock prices provided in the ‘task2\_stock\_data.xlsx’ excel attachment. You can use any standard packages available for this purpose. To reduce outliers please implement winsorization on the returns before using the data in beta calculation. You can use the [following function](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mstats.winsorize.html) to winsorize the data with a 5% limit on both right and left tails.

The beta calculation should use ‘as of date’, ‘beta calculation window’ and ‘observation frequency’ as inputs. For example: we should be able to calculate beta(s) for as of date = ‘2021/10/31’, window = ‘1y’ and observation frequency = ‘weekly’ returns for the data shared in the file. Other inputs for the ‘frequency’ parameter can be: Daily, Weekly, Bi-weekly, Monthly or Quarterly.

Expected output from the function would be a list or dictionary of beta value corresponding to each stock ticker in the data.Please provide us your output for the following scenarios:

asofdate=2021-10-31, window=1y, frequency=daily

asofdate=2021-10-31, window=1y, frequency=weekly

**Input data description:**

We are sharing stock prices data for stock tickers along with the associate SPX (SPY) index's total returns in the ’task2\_stock\_data.xlsx’ attachment.