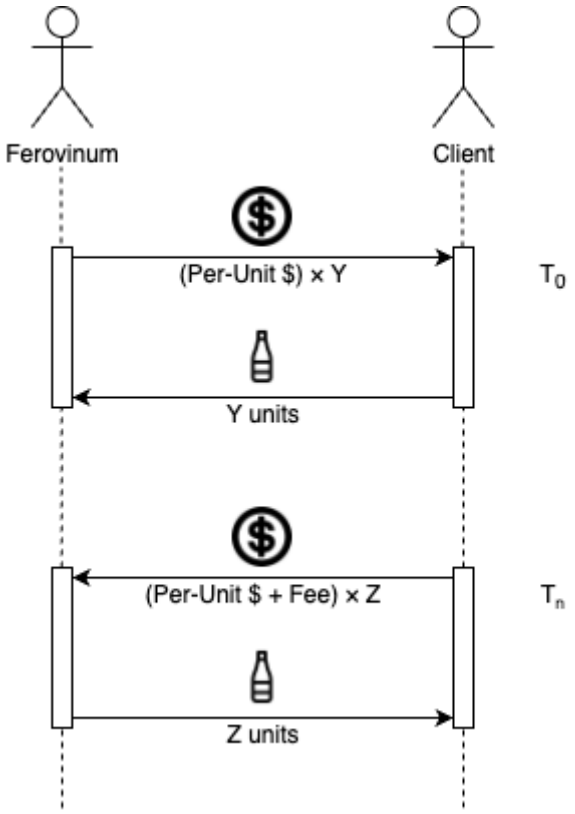
Financial Transaction Engine

Overview

Being a fast growing FinTech, Ferovinum's core product provides easily accessible and granular funding for its clients' wine and/or spirits products. The activity between Ferovinum and its clients are represented by financial transactions which can be simplified buying/selling of products.

For the purpose of this exercise, a simplified round-trip engagement between Ferovinum and its client would be as follows:

1. Client sells Y bottles of wine to Ferovinum
2. Ferovinum buys Y bottles of wine from client at predetermined amount; the per-unit price is solely based on the particular product
3. Time has passed, the client buys Z bottles of the same wine from Ferovinum; note that $Z \leq Y$
4. Ferovinum sells Y bottles of same wine to client at predetermined amount with fee; the fee rate are solely based on the particular client and is expressed in annualised terms compounded monthly (partial months are considered full months, compounding starts from day 0, i.e. 1 month of fee will incur upon Ferovinum buying the stock); stock is depleted on a first-in-first-out (FIFO) basis



For the above example, if the per-unit price was $10, the fee was 10% p.a. and T_n was 1 year away, the per-unit price at T_n would be:

$$10 \times (1 + \frac{0.1}{12})^{12} = 11.05$$

There are some assumptions/restrictions that Ferovinum must follow:

1. The predetermined per-unit price is solely based on the product alone and does not change over time
2. The fee rate is unique to each client and does not change over time
3. Ferovinum cannot be in a short position for any product at any time, i.e. cannot sell products it does not already have
4. Conversely, clients cannot buy products that they did not previously sell to Ferovinum
5. Ferovinum cannot sell products to a client that was different to the one that the product was bought from, i.e. the client that Ferovinum initial bought from must be the same client to sell to for each particular product

Task

You will need to create a simple HTTP server that accepts buy/sell orders, persist the transactions in a local database and implement data query endpoints to extract information from the transactions.

You can implement your solution in any programming language of your choice. We have provided an OpenAPI specification where you can use Swagger Editor to generate code for a stub server in various programming languages, see example Python stub server.

We will interact with your simple HTTP server via Postman, all endpoints specified in the OpenAPI specification will need to be implemented with expected behaviour for prescribed success/failure responses.

As described in the simplified round-trip engagement above, we also provide the predetermined product per-unit price and the client fee which are used for all your calculations.

Endpoints

Consider the following orders to be processed by the simple HTTP server and will be used as an example to describe the endpoints required to be implemented for this exercise.

timestamp	type	clientId	productId	quantity
2020-01-01T10:00:00Z	buy	C-1	P-1	1000
2020-01-01T10:00:00Z	buy	C-2	P-3	2000
2020-01-01T10:00:00Z	buy	C-1	P-2	500
2020-01-01T11:00:00Z	sell	C-1	P-1	50
2020-02-01T10:00:00Z	sell	C-1	P-1	100
2020-02-28T10:00:00Z	sell	C-2	P-3	100
2020-06-15T10:00:00Z	sell	C-1	P-2	250
2020-12-01T10:00:00Z	buy	C-3	P-1	5000
2021-01-01T10:00:00Z	sell	C-2	P-3	1900
2021-07-01T10:00:00Z	sell	C-3	P-1	1
2022-01-01T10:00:00Z	sell	C-1	P-2	249
2022-12-01T10:00:00Z	sell	C-3	P-1	4999

`POST /order`

Process buy/sell orders where stock balance is increased/decreased accordingly. Each record in the above example represents unique `OrderRequest` payloads that are used for this endpoint.

The corresponding `OrderRequest` content for the above example is as follows; note that all calculations are based on the provided predetermined product per-unit price and the client fee and rounded to 2 decimal places.

clientId	productId	price
C-1	P-1	47.90
C-2	P-3	7.99
C-1	P-2	36.18
C-1	P-1	49.46
C-1	P-1	51.06
C-2	P-3	8.31
C-1	P-2	43.83
C-3	P-1	47.90
C-2	P-3	10.34
C-3	P-1	57.23
C-1	P-2	80.49
C-3	P-1	83.54

`GET /balance/client/{clientId}?date={date}`
List all product quantities for given `clientId`. The optional `date` parameter specifies a particular date to extract a snapshot past result, otherwise the latest result will be returned.

Using the above example, the following requests should have the expected response:

- *`GET /balance/client/C-1`*
  `[{"clientId": "C-1", "productId": "P-1", "quantity": 850},`
  `{"clientId": "C-1", "productId": "P-2", "quantity": 1}]`
- *`GET /balance/client/C-1?date=2020-01-15`*
  `[{"clientId": "C-1", "productId": "P-1", "quantity": 950},`
  `{"clientId": "C-1", "productId": "P-2", "quantity": 500}]`

`GET /balance/product/{productId}?date={date}`
List all product quantities for given `productId`. The optional `date` parameter specifies a particular date to extract a snapshot past result, otherwise the latest result will be returned.

Using the above example, the following requests should have the expected response:

- *`GET /balance/product/P-1`*
  `[{"clientId": "C-1", "productId": "P-1", "quantity": 850},`
  `{"clientId": "C-3", "productId": "P-1", "quantity": 0}]`
- *`GET /balance/product/P-1?date=2021-07-01`*
  `[{"clientId": "C-1", "productId": "P-1", "quantity": 950},`
  `{"clientId": "C-3", "productId": "P-1", "quantity": 4999}]`

`GET /portfolio/client/{clientId}?date={date}`
Calculate portfolio metrics for given `clientId`. The optional `date` parameter specifies a particular date to extract a snapshot past result, otherwise the latest result will be returned.

You should be calculating metrics:

- `lifeToDateFeeNotional`: Notional amount of all fees ever charged for client
- `lifeToDateProductNotional`: Notional amount of all products ever transacted with client
- `outstandingFeeNotional`: Notional amount of outstanding fees with client
- `outstandingProductNotional`: Notional amount of outstanding products for client
- `weightedAverageRealisedAnnualisedYield`: Annualised yield of fees weighted on notional amount of sold stock
- `weightedAverageRealisedDuration`: Average holding duration (days) weighted on notional amount of sold stock

Using the above example, the following requests should have the expected response:

- *`GET /portfolio/client/C-1?date=2024-01-01`*
  `{``"lifeToDateFeeNotional"`*`: 13339.69,`*
  `"lifeToDateProductNotional"`*`: 659900,`*
  `"outstandingFeeNotional"`*`: 154573.73,`*
  `"outstandingProductNotional"`*`: 40751.18,`*

"weightedAverageRealisedAnnualisedYield": 11416.76,
"weightedAverageRealisedDuration": 327.30*}*

GET /transactions/client/{clientId}?fromDate={fromDate}&toDate={toDate}

List all transactions for given `clientId`. The optional `fromDate, toDate` parameters specifies a particular date range to filter results, otherwise all transactions will be returned.

Using the above example, the following requests should have the expected response:

- *GET /transactions/client/C-1*
  [{"clientId": "C-1", "productId": "P-1", "orderType": "buy", "price": 47.9, "quantity": 1000, "timestamp": "2020-01-01T10:00:00Z"},
  {"clientId": "C-1", "productId": "P-2", "orderType": "buy", "price": 36.18, "quantity": 500, "timestamp": "2020-01-01T10:00:00Z"},
  {"clientId": "C-1", "productId": "P-1", "orderType": "sell", "price": 49.46, "quantity": 50, "timestamp": "2020-01-01T11:00:00Z"},
  {"clientId": "C-1", "productId": "P-1", "orderType": "sell", "price": 51.06, "quantity": 100, "timestamp": "2020-02-01T10:00:00Z"},
  {"clientId": "C-1", "productId": "P-2", "orderType": "sell", "price": 43.83, "quantity": 250, "timestamp": "2020-06-15T10:00:00Z"},
  {"clientId": "C-1", "productId": "P-2", "orderType": "sell", "price": 80.49, "quantity": 249, "timestamp": "2022-01-01T10:00:00Z"}]
- *GET /transactions/client/C-3?fromDate=2021-01-01&romDate=2023-01-01*
  [{"clientId": "C-3", "productId": "P-1", "orderType": "sell", "price": 57.23, "quantity": 1, "timestamp": "2021-07-01T10:00:00Z"},
  {"clientId": "C-3", "productId": "P-1", "orderType": "sell", "price": 83.54, "quantity": 4999, "timestamp": "2022-12-01T10:00:00Z"}]

GET /transactions/product/{productId}?fromDate={fromDate}&toDate={toDate}

List all transactions for given `productId`. The optional `fromDate, toDate` parameters specifies a particular date range to filter results, otherwise all transactions will be returned.

Using the above example, the following requests should have the expected response:

- *GET /transactions/product/P-1*
  [{"clientId": "C-1", "productId": "P-1", "orderType": "buy", "price": 47.9, "quantity": 1000, "timestamp": "2020-01-01T10:00:00Z"},

```
{"clientId": "C-1", "productId": "P-1", "orderType": "sell",
"price": 49.46, "quantity": 50, "timestamp":
"2020-01-01T11:00:00Z"},
{"clientId": "C-1", "productId": "P-1", "orderType": "sell",
"price": 51.06, "quantity": 100, "timestamp":
"2020-02-01T10:00:00Z"},
{"clientId": "C-3", "productId": "P-1", "orderType": "buy",
"price": 47.9, "quantity": 5000, "timestamp":
"2020-12-01T10:00:00Z"},
{"clientId": "C-3", "productId": "P-1", "orderType": "sell",
"price": 57.23, "quantity": 1, "timestamp":
"2021-07-01T10:00:00Z"},
{"clientId": "C-3", "productId": "P-1", "orderType": "sell",
"price": 83.54, "quantity": 4999, "timestamp":
"2022-12-01T10:00:00Z"}]
```


- *GET /transactions/product/P-2?fromDate=2021-01-01&romDate=2023-01-01*


```
[{"clientId": "C-1", "productId": "P-2", "orderType": "sell",
"price": 80.49, "quantity": 249, "timestamp":
"2022-01-01T10:00:00Z"}]
```


Deliverables

- Your source code and test input data in a private Github repository
- A README file that outlines:
  - How to compile/run your command-line program
  - Brief description of the code structure
  - Any limitations of your implementation (or other things you'd like to point out)
  - Any trade-offs or design decisions you made that are worth noting
- You will have 24 hours to complete the task
- The above test scenario is just an example - as part of this exercise, you are required to thoroughly test your application yourself
- Your solution should be able to handle up to 100 unique clients, 1,000 unique products and 1,000,000 orders - you can expect orders from client will not exceed 5 per day i.e. not high frequency trading environment

Tips

- Do not over engineer your solution. We are looking for simple solutions and clearly written code
- Do not rush, do not write spaghetti code
- Specific details about the behaviour of the application are left for you to decide how best to handle