Bad Broker Test Task

Overview

Let's imagine that you are a trader with a certain amount of USD and you can buy other currencies through your broker. So your job is to try to predict currencies fluctuations and make money. You can only open one position (buy and sell) within a period. In this test task you should determine what would be the best revenue for a specified historical period. It can be used for statistic purposes. From a high level of view you need to implement function:

```
f(period rates array, start date, end date, USD) = (currency, buy date, sell date, revenue USD)
```

Business Requirements

1. You have only one chance to buy and sell within a specified period.
2. You can only buy RUB, EUR, GBP and JPY. You can only sell USD.
3. Your broker is not good and requires a $1 fee for each opened position day (the day when you keep your money in non-USD currency after buying and before selling).
4. You rely on long-term trade only. We only take into account the official daily exchange rate.
5. The specified historical period cannot exceed 2 months (60 days).

Technical Requirements

1. Microsoft Visual Studio 2019 compatible solution.
2. It should be ASP.NET Core API application. Use Swagger to make API testing much more convenient.
3. There should be `rates/best` endpoint with parameters `startDate`, `endDate`, and `moneyUsd`. See example below.
4. To get rates use one of these services:
   - https://exchangeratesapi.io/
   - https://openexchangerates.org/
   - https://fixer.io/
   - https://openrates.io/
5. Use English for application UI and code comments.
6. All business logic and data loading should be done in backend using C# language.
7. We expect that you will implement the solution in the object-oriented approach. For example, we need separate classes to load rates, for calculations and rate models. The logic should be out of web controllers. Exceptions handling should be implemented as well.
8. Unit tests.
9. *(Bonus)* The application should cache retrieved rate data to the database and download rates for any other dates on demand. For instance, if a user selects data from 01/01 to 01/31 and then from 01/15 to 02/15 part of data will be retrieved from the database (01/15 to 01/31) whereas the other part (02/01 to 02/15) will be from external service.
10. *(Bonus)* prepare frontend solution using Vue.js.

Example 1

Date	***RUB/USD***
2014-12-15	60.17
2014-12-16	72.99
2014-12-17	66.01
2014-12-18	61.44
2014-12-19	59.79
2014-12-20	59.79
2014-12-21	59.79
2014-12-22	54.78
2014-12-23	54.80

You have $100 and the specified period is December 2014. The best case for this period is RUB for 12/16/2014-12/22/2014 . If you bought RUB on 12/16/2014 and sell them 12/22/2014 you would get ~$127. So revenue is $27.

`(72.99 * 100 / 54.78) - 6 (days broker fee) = $127.24`.

Example 2

You have $50 and history data:

Date	***value/USD***
2012-01-05	40.00
2012-01-07	35.00
2012-01-19	30.00

The revenue for these periods is:

```
* 2012-01-05 - 2012-01-07: (40 * 50 / 35) - (7 - 5) = 55.6
```

```
* 2012-01-07 - 2012-01-19: (35 * 50 / 30) - (19 - 7) = 49.3
```

```
* 2012-01-05 - 2012-01-19: (40 * 50 / 30) - (19 - 5) = 51.6
```

So the best dates are 01/05/2012 and 01/07/2012!

Example Request and Response

```
curl -X GET
"http://localhost:5000/rates/best?startDate=2014-12-15&endDate=2014-12-23&moneyUsd=100" -H
"accept: application/json"
```

```
{
  "rates": [
    {
      "date": "2014-12-15T00:00:00",
      "rub": 60.1735876388,
      "eur": 0.8047642041,
      "gbp": 0.6386608724,
      "jpy": 118.8395300177
    },
    // ...
  ],
  "buyDate": "2014-12-16T00:00:00",
  "sellDate": "2014-12-22T00:00:00",
  "tool": "RUB",
  "revenue": 27.258783297622983
}
```