A Forex company is looking to partner with an external broker for executing trades on behalf of its clients. The main requirement is to create a new Spring Boot microservice that will expose a set of REST APIs for submitting trade requests as well as retrieving details about the statuses of the requests. Following is a description of the APIs that need to be exposed to its clients.

API Description:

Submit Buy/Sell trade request

APIs for submitting a buy or sell trade request

Request:

symbol: The symbol to execute. Can be either `USD/JPY` or `EUR/USD`.

Invalid symbols must be rejected with a `400 - Bad Request` http code and along with a validation error message i.e. *"Symbol valid values: USD/JPY, EUR/USD"*

quantity: Integer indicating the quantity to execute. Must be `0 < quantity < 1M`.

Invalid values must be rejected with a `400 - Bad Request` http code along with a validation error message i.e. "*quantity must be greater than 0 and less than or equal to 1M"*

price: The price the trade will be executed on. Must be a positive nonzero decimal number.

Invalid values must be rejected with a `400 - Bad Request` http code and proper error message i.e. "price must be greater than 0"

Response

For successful request server must response with a 201 – CREATED `status code` along with `Location` header that is the API for retrieving the requested trade status

Sample request

POST http://localhost:8080/api/buy

```
{
    "symbol": "EUR/USD",
    "quantity": 1000,
    "price": 1.123
}
```

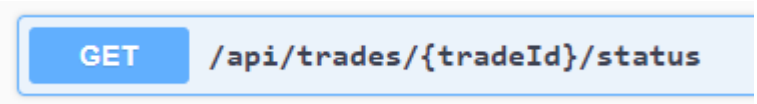
Sample response

HTTP Status: 201 - CREATED

Location: http://localhost:8080/api/trades/2b42f60f-c794-43d8-b4a3-da709f3d2fa6/status

Get trade status

API for retrieving the trade status based on the id.



Response

status: PENDING_EXECUTION, EXECUTED or NOT_EXECUTED

Sample request

GET http://localhost:8080/api/trades/2b42f60f-c794-43d8-b4a3-da709f3d2fa6/status

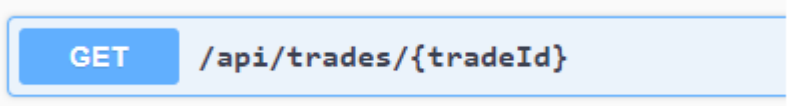
Sample response body

HTTP Status: 200 - OK

```
{
    "status": "NOT_EXECUTED"
}
```

Get trade details

API for retrieving trade details by id



Response:

id: The trade id

quantity: The trade quantity

side: BUY or SELL

price: The price

status: Trade status

reason: Unsuccessful trades reason text (or empty if trade is successful)

timestamp: trade submission timestamp

Sample request

```
GET http://localhost:8080/api/trades/2b42f60f-c794-43d8-b4a3-da709f3d2fa6
```

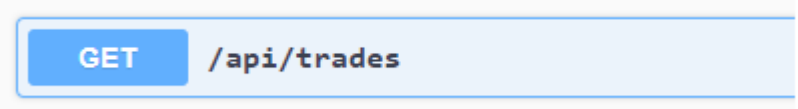
Sample response

```
HTTP Status: 200 - OK
```


```
{
    "id": "2b42f60f-c794-43d8-b4a3-da709f3d2fa6",
    "quantity": 1000,
    "symbol": "EUR/USD",
    "side": "BUY",
    "price": 1.123,
    "status": "PENDING_EXECUTION",
    "reason": null,
    "timestamp": "2022-02-14T17:55:05.68645"
}
```


Get all trades

API for retrieving all trades



Response:

Return a list of all submitted trade requests

Sample request

```
GET http://localhost:8080/api/trades
```

Sample response

```
HTTP Status: 200 - OK
```


```
[
    {
        "id": "2b42f60f-c794-43d8-b4a3-da709f3d2fa6",
        "quantity": 1000,
        "symbol": "EUR/USD",
        "side": "BUY",
        "price": 1.123,
        "status": "PENDING_EXECUTION",
        "reason": null,
        "timestamp": "2022-02-14T12:19:31.715037"
    },
```

```
    {
        "id": "e22ea403-79cc-403f-869d-8f3481e8582d",
        "quantity": 10000000,
        "symbol": "USD/JPY",
        "side": "SELL",
        "price": 1.00,
        "status": "NOT_EXECUTED",
        "reason": "No available quotes",
        "timestamp": "2022-02-14T17:22:52.27468"
    },
    {
        "id": "259f6875-65b5-4e6d-a630-0780bb8c5595",
        "quantity": 10000000,
        "symbol": "USD/JPY",
        "side": "SELL",
        "price": 1.12,
        "status": "EXECUTED",
        "reason": null,
        "timestamp": "2022-02-14T17:36:13.00751"
    },
    {
        "id": "8494e1af-58b8-4270-b230-cb9838d596bc",
        "quantity": 1000,
        "symbol": "EUR/USD",
        "side": "BUY",
        "price": 1.12,
        "status": "NOT_EXECUTED",
        "reason": "trade expired",
        "timestamp": "2022-02-14T17:55:05.68645"
    }
]
```


Business Logic:

1. Once a trade request is submitted, the server responds immediately with `201 – CREATED` indicating that it has accepted the request and **takes full responsibility to execute it** on the external broker. The response will contain a URL location to retrieve the status of the trade.
2. The initial status of the trade after the request is accepted must be `PENDING_EXECUTION`. You may use an embedded DB of your choice to store any information you might need.
3. External Broker source code is provided (see next section). It provides some dummy classes to simulate integration with external broker.
4. To simulate network errors and service crashes the External Broker module might not send any response at all. If no response is received within 1 minute the trade can be considered as NOT_EXECUTED with reason “trade expired”.

Provided source code

The following files are included in the `com.broker.external package`:

`ExternalBroker`: Simulates integration with the external broker. It accepts `BrokerTrade` as parameter and asynchronously invokes `BrokerResponseCallback` when there is a trade update. The response can be either successful or unsuccessful.

`BrokerResponseCallback`: Callback interface for handling trade response

`BrokerTrade`: Data transfer Object class with trade fields accepted by the external broker

`BrokerTradeSide`: Trade side `enum` class. Can be either `BUY` or `SELL`.

Instructions:

- Provided classes in `com.broker.external` **cannot be changed**.
- You must provide test cases that prove your code behaves as expected
- Add logging where you feel it's appropriate
- Try to spend **no more than 3 to 4 hours** on this exercise