Backend Developer Assignment

Introduction

The BUX mobile apps communicate with the BUX backend through an HTTP RESTful API and messages exchanged over a WebSocket connection.

The data format of the REST API is structured in JSON, as well as the messages exchanged over WebSocket.

On top of the WebSocket connection, we have created an application protocol based on the concept of real-time feed "channels". The client can subscribe to these channels in order to receive real-time updates for it.

We leverage the 'full-duplex' nature of WebSockets: the client has to send a WebSocket message to subscribe for a channel, and from this moment on, until unsubscription (or disconnect), he will start receiving messages with updates on this channel over the WebSocket connection.

Assignment Requirements

The goal of this assignment is to build a very basic Trading Bot that tracks the price of a certain product and will execute a pre-defined trade in the said product when it reaches a given price. After the product has been bought the Trading Bot should keep on tracking the prices and execute a sell order when a certain price is hit. In the input, there should be an "upper limit" price and a "lower limit" price.

At startup the Trading Bot should gather four inputs;

1. The product id (see below for a suggested list to test with)
2. The buy price. If the stock price doesn't reach that price the position shouldn't be opened.
3. The upper limit sell price. This is the price you are willing to close a position and make a profit.
4. The lower limit sell price. This the price you want are willing to close a position at and make a loss.

Note that for the trading logic to be valid, the relation between the buy price and the lower / upper limit should be: lower limit sell price < buy price < upper limit sell price. Think about what it means to close a position with a profit. What should the relation between the current price and the upper limit selling price should be when deciding to close the position or not?

The Trading Bot should then subscribe to the Websocket channel for the given product id and when the buy price is reached it should execute the buy order (API definition below) and then when one of the limits is reached it should execute the sell order

Observations

1. Feel free to ask questions if necessary.
2. **Important**: please note that the Markets aren't opened during weekends, so you won't be able to get trade prices real-time updates during most of the time during weekends for almost all products. Please check some of the products opening hours: https://support.getbux.com/en/support/solutions/articles/1000119518-what-are-the-opening-hours-of-all-products-. To accommodate this, we have created a simplified version of the BUX backend that can be run standalone. This standalone backend will simulate price quotes on products and is not limited to any opening hours. It is a JAR file called *bux-server.jar* and can be run with the command *java -jar bux-server.jar*. A download link to this file should have been provided to you. The standalone BUX backend server requires the Java JDK to run, which can be downloaded here: http://www.oracle.com/technetwork/java/javase/downloads/index.html. The standalone server runs on http://localhost:8080. For all mentioned endpoint URLs you can replace https://api.beta.getbux.com with http://localhost:8080.
3. **Important:** please keep in mind that the standalone BUX backend server is meant as a supporting tool for development only. Although it mimics the functionality of our BETA environment, it can never fully replace it. It is functional for all intents and purposes of the assignment, but does not provide comprehensive data validation nor API robustness. The completed assignment should therefore always point to the BETA environment.
4. You can use any 3rd party library you like.

API and real-time channel specification

Some product ids you can use for testing (in respective order these are ids for products labeled as 'Germany30', 'US500', 'EUR/USD', 'Gold', 'Apple' and 'Deutsche Bank'):

- sb26493
- sb26496
- sb26502
- sb26500

- sb26513
- sb28248

WebSocket connection and handling events

In order to connect to the WebSocket real-time feed, we have established a confirmation on the application protocol.

After you have successfully connected to the WebSocket, our backend will generate a "connect.connected" event as a result of a successful connection.

```
{
    "t": "connect.connected",
    "body": {
   ...
        }
    }
}
```

You should **only proceed with subscriptions after receiving this event**.

For various reasons, the connection at the application level can also fail and this event will be generated (this is just an example of an error):

```
{
    "t": "connect.failed",
    "body": {
        "developerMessage": "Missing JWT Access Token in request",
        "errorCode": "RTF_002"
    }
}
```

There are some more events the clients can receive for which they don't have to subscribe to. For the sake of this assignment, you can just ignore them and handle only the product quote event.

WebSocket real-time feed, Products Channel

WEBSOCKET URL

https://rtf.beta.getbux.com/subscriptions/me

You also have to use the following Request Headers when connecting in order to be authorized:

```
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJyZWZyZXNoYWJsZSI6ZmFsc2UsInN1YiI6ImJiMGNkYTJiLWE
xMGUtNGVkMy1hZDVhLTBmODJiNGMxNTJjNCIsImF1ZCI6ImJldGEuZ2V0YnV4LmNvbSIsInN
jcCI6WyJhcHA6bG9naW4iLCJydGY6bG9naW4iXSwiZXhwIjoxODIwODQ5Mjc5LCJpYXQiOjE
1MDU0ODkyNzksImp0aSI6ImI3MzlmYjgwLTM1NzUtNGIwMS04NzUxLTMzZDFhNGRjOGY5MiI
sImNpZCI6Ijg0NzM2MjI5MzkifQ.M5oANIi2nBtSfIfhyUMqJnex-JYg6Sm92KPYaUL9GKg
Accept-Language: nl-NL,en;q=0.8
```

SUBSCRIPTION MESSAGE

To change subscription for quote updates send the following message:

```
{
   "subscribeTo": [
      "trading.product.{productId}"
   ],
   "unsubscribeFrom": [
      "trading.product.{productId}"
   ]
}
```

CHANNEL UPDATES

The messages you receive through the WebSocket have a common structure, pictured below with a "trading.quote" example:

```
{
   "t": "trading.quote",
   "body": {
      "securityId": "{productId}",
      "currentPrice": "10692.3"
   }
}
```

t means the type of the event. You should always check it and relate to the subscription you're interested in, as you can receive multiple different events.

body is the content of the event, which varies depending on the type of the event being received

Trade API calls

All calls to the BUX API need to have the following headers:

```
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJyZWZyZXNoYWJsZSI6ZmFsc2UsInN1YiI6ImJiMGNkYTJiLWE
xMGUtNGVkMy1hZDVhLTBmODJiNGMxNTJjNCIsImF1ZCI6ImJldGEuZ2V0YnV4LmNvbSIsInN
jcCI6WyJhcHA6bG9naW4iLCJydGY6bG9naW4iXSwiZXhwIjoxODIwODQ5Mjc5LCJpYXQiOjE
1MDU0ODkyNzksImp0aSI6ImI3MzlmYjgwLTM1NzUtNGIwMS04NzUxLTMzZDFhNGRjOGY5MiI
sImNpZCI6Ijg0NzM2MjI5MzkifQ.M5oANIi2nBtSfIfhyUMqJnex-JYg6Sm92KPYaUL9GKg
Accept-Language: nl-NL,en;q=0.8
Content-Type: application/json
Accept: application/json
```

Make a Trade (Buy order)

To make a trade the following data needs to be sent as a HTTP POST to *https://api.beta.getbux.com/core/21/users/me/trades*

```
{
    "productId" : "sb26493",
    "investingAmount" : {
        "currency": "BUX",
        "decimals": 2,
        "amount": "10.00"
    },
    "leverage" : 2,
    "direction" : "BUY",
    "source": {
        "sourceType": "OTHER"
    }
}
```

If everything goes ok you will receive the following data back

```
{
    "id": "98922f1a-4c10-4635-a9e6-ae19ddcd12b4",
    "positionId": "4c58a0b2-ea78-46a0-ac21-5a8c22d527dc",
    "product": {
        "securityId": "{productId}",
        "symbol": "{productSymbol)",
        "displayName": "{productName}"
    },
    "investingAmount": {
        "currency": "BUX",
        "decimals": 2,
        "amount": "200.00"
    },
    "price": {
        "currency": "EUR",
        "decimals": 3,
        "amount": "0.567"
    },
    "leverage": 1,
    "direction": "BUY",
    "type": "OPEN",
    "dateCreated": 1405515165705
}
```


The most important piece of data here is the positionId, you need it later to close the position (i.e. place the sell order)

For all the details on this API method, see the attached document

Close Position (sell order)

To close a position the following url needs to be called (with an empty request body) as a HTTP DELETE to *https://api.beta.getbux.com/core/21/users/me/portfolio/positions/{positionId}*

If everything goes ok you will receive the following data back

```
{
  "id": "ebd37d2b-8489-4419-bd78-8df7dc6a4823",
  "positionId": "423ac625-bd9a-41eb-8531-d5ea25352020",
  "profitAndLoss": {
    "currency": "BUX",
    "decimals": 2,
    "amount": "-0.61"
  },
  "product": {
    "securityId": "sb26493",
    "symbol": "GERMANY30",
    "displayName": "Germany 30"
  },
  "investingAmount": {
    "currency": "BUX",
    "decimals": 2,
    "amount": "200.00"
  },
  "price": {
    "currency": "EUR",
    "decimals": 1,
    "amount": "10342.5"
  },
  "leverage": 2,
  "direction": "SELL",
  "type": "CLOSE",
  "dateCreated": 1473946766125
}
```


For all the details on this API method, see the attached document