Mobile 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.

Given the described environment, in this small assignment we want to see how you deal, integrated with an user interface and keeping it consistent, with the issues that arise while handling connections, threading, providing user feedback, performing requests, treating errors and such.

Assignment Requirements

The main goal is to build a very simple client app that does the following:

1. In a first screen, obtain, from the user input or from a list of pre-defined items, a Product Identifier; what we call a "Product" is a *Trading product*, such as a stock from Apple or Facebook.
2. In a second screen, show the data of the Product with the provided Identifier.
    - To show current price of the Product, please take into account the right decimal digits and locale-aware currency formatting.
    - Besides the value of the previous day closing price, please also show the difference of the previous day closing price to the current price, in %. (e.g. if previous day = $100,00 and current = $150,00, the % difference will be 50%).
3. Subscribe to the real-time feed channel of this Product.
4. Update the information shown according to the WebSocket update messages (i.e. updated price value and previous day % difference).

Remember to also properly handle common scenarios and errors from the nature of a mobile app, such as an unstable mobile connection, app switching, disconnection, etc. It's up to you to decide what to do in these cases.

Observations

- The final project should be simple enough for the scope of the assignment (don't over-engineer!), but solid in good engineering principles and attention to details.
- It is fine to cut corners, as long as they don't damage your overall solution and you're able to justify them and present alternatives.
- Feel free to ask questions if necessary.
- You can use any 3rd party library you like.
- **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: http://en.support.getbux.com/support/solutions/articles/1000119518-what-are-the-opening-hours-of-all-products-. To accommodate this, we have created a mock version of the BUX backend that can be run standalone, check the following section for details.

Standalone mock backend

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, a* dow nload link to this file should have been provided to you.

The standalone mock backend server requires the Java JDK to run, which can be downloaded at http://www.oracle.com/technetwork/java/javase/downloads/index.html. The standalone server runs on http://localhost:8080.

To use:

- Run it with the command: *java -jar bux-server.jar*
- For all mentioned endpoint URLs replace https://api.beta.getbux.com and https://rtf.beta.getbux.com with http://localhost:8080

Please keep in mind that the standalone mock backend server is meant as a supporting tool for development only. Although it mimics the functionality of our backend, 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 real BETA backend.

API and real-time channel specification

Some product ids you can use for testing:

- Germany30: sb26493
- US500: sb26496
- EUR/USD: sb26502
- Gold: sb26500
- Apple: sb26513
- Deutsche Bank: sb28248

RESTful Get Product Details

HTTP GET URL

https://api.beta.getbux.com/core/23/products/{productId}

Use the following Request Headers:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.
eyJyZWZyZXNoYWJsZSI6ZmFsc2UsInN1YiI6ImJiMGNkYTJiLWExMGUtNGVkMy1hZDVhLTBm
ODJiNGMxNTJjNCIsImF1ZCI6ImJldGEuZ2V0YnV4LmNvbSIsInNjcCI6WyJhcHA6bG9naW4i
LCJydGY6bG9naW4iXSwiZXhwIjoxODIwODQ5Mjc5LCJpYXQiOjE1MDU0ODkyNzksImp0aSI6
ImI3MzlmYjgwLTM1NzUtNGIwMS04NzUxLTMzZDFhNGRjOGY5MiIsImNpZCI6Ijg0NzM2MjI5
MzkifQ.M5oANIi2nBtSfIfhyUMqJnex-JYg6Sm92KPYaUL9GKg
Accept: application/json
Accept-Language: nl-NL,en;q=0.8
```

RESPONSE (MAIN DATA TO SHOW):


```
{
   "symbol": "FRANCE40",
   "securityId": "26608",
   "displayName": "French Exchange",
   "currentPrice": {
      "currency": "EUR",
      "decimals": 1,
      "amount": "4371.8"
   },
   "closingPrice": {
      "currency": "EUR",
      "decimals": 1,
      "amount": "4216.4"
   }
}
```

- **securityId** is the Product ID.
- **closingPrice** means the previous day closing price of the Product.
- Currency codes are in the ISO 4217 format.

HTTP ERRORS:

In case of an error 4xx or 5xx HTTP status code is returned.

The following object may be returned in the body to describe an error:

```
{
    "message": "may be null",
    "developerMessage": "technical description of the error",
    "errorCode": "AUTH_001"
}
```

As an additional clarification, here are some of the possible values of the field **errorCode**:

BUX Error Code	Possible reason
TRADING_002	unexpected error
AUTH_007	access token is not valid
AUTH_014	user does not have sufficient permissions to perform this action
AUTH_009	missing Authorization header
AUTH_008	access token is expired

WebSockets 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 authorised:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.
eyJyZWZyZXNoYWJsZSI6ZmFsc2UsInN1YiI6ImJiMGNkYTJiLWExMGUtNGVkMy1hZDVhLTBm
ODJiNGMxNTJjNCIsImF1ZCI6ImJldGEuZ2V0YnV4LmNvbSIsInNjcCI6WyJhcHA6bG9naW4i
LCJydGY6bG9naW4iXSwiZXhwIjoxODIwODQ5Mjc5LCJpYXQiOjE1MDU0ODkyNzksImp0aSI6
ImI3MzlmYjgwLTM1NzUtNGIwMS04NzUxLTMzZDFhNGRjOGY5MiIsImNpZCI6Ijg0NzM2MjI5
MzkifQ.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.

WEBSOCKET LIBRARIES

We'd like to throw in a few suggestions for libraries to be used to handle the WebSocket on mobile. Note that the list has no particular order and you should pick whatever suits your project/style most (whether it is in the list or not):

- Android
    - OkHttp Web Sockets: https://github.com/square/okhttp/
    - Scarlet: https://github.com/Tinder/Scarlet
    - AndroidAsync: https://github.com/koush/AndroidAsync
    - Java-WebSocket: https://github.com/TooTallNate/Java-WebSocket
    - nv-websocket-client: https://github.com/TakahikoKawasaki/nv-websocket-client
    - Tyrus: https://github.com/eclipse-ee4j/tyrus
- iOS
    - https://github.com/daltoniam/Starscream
    - https://github.com/acmacalister/jetfire
    - https://github.com/tidwall/SwiftWebSocket
    - https://developer.apple.com/documentation/foundation/urlsessionwebsockettask (Native)

[iOS] Since there are no restrictions in the minimum deployment target for iOS, feel free to use SwiftUI