NFTNet Project Report

This project aims to allow a client to communicate with a server to request some data where the communication in between is ruled by a custom application layer protocol. The server, acting like the middle element between “coingecko.com” and the client, takes requests from the client and searches for the requested data in “coingecko.com” using it’s API and returns that one to the client.

CoinGeckoProtocol (CGP)

The application layer protocol, namely “CoinGeckoProtocol (CGP)” is designed to enable both client and server side to easily construct and interpret the messages. The message structure includes various fields, which are all enum types except the body field. This allows standardized communication between the ends. You can see the message fields in Figure 1. The “Type” field

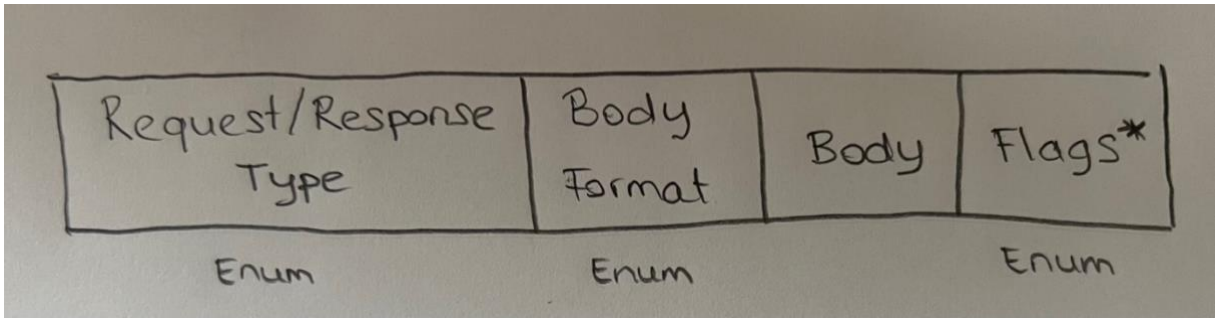


Figure 1

for request and response are different as expected. The “Body Format” and “Body” fields are shared, and “Flags” are used by the client to construct the message and used by server to interpret it. The delimiter between the parameters is “|||”. Therefore, before the body is appended to the message, first the actual occurrences (if exist) of “|||” is replaced by a placeholder which is again replaced by the original one at the other end. Additionally, each new line character is also replaced by another placeholder when sending the message to the other end. This one is also restored at the other end.

The protocol package provides some functionalities for both the server and client. It allows them to create messages by specifying fields and converting the messages to the appropriate format for transmission (replacing with placeholder etc.). Also, when a message is received, it is parsed with the help of the protocol utilities. These properties, acting as an interface, allows

the abstraction of inner processes such as the necessary modifications for transmission and restoring the original meaningful message.

Project Overview

There are 3 modules in Server project and 2 in Client project. Both projects contain the protocol package. The server contains the Server package where resides the server specific classes accepting new clients to communicate by creating a new thread for each, handling the communication with CoinGecko and managing the response logic based on the request messages. In client side, there exist the client code, providing an instructive and informative textual interface for the users, and handling the initialization and management of the communication with server. At the server side, there exist an additional module named “CoinGecko”, which is an interface facilitating the usage of the CoinGecko API.

Project Specifications

Server Side

In the “Server” package, in addition to the main class “RunServer” where the server address, server port and timeout for a given communication are specified, there exist two classes: “GeckoServer” is the class of the object responsible for handling the incoming connection requests. It runs in a while loop to welcome new clients by creating a socket for the communication and initialize the “GeckoServerThread” object with that socket. The “GeckoServerThread” class is responsible for the managing the communication per client. Therefore, it is also the one who communicates with the CoinGecko API. The latter is achieved by an adapter class named “CoinGeckoAdapter”, which is implemented with Singleton pattern to be used by all threads. It provides 2 functions “queryList()” returning the list of available NFTs and “queryNFT(nftID)” returning information about a given NFT. If an error occurs, these functions throw a custom exception named “GeckoAPIException” which should be handled by the “GeckoServerThread”. To handle the communication, “GeckoServerThread” object runs in a loop. It waits for a request from the client, parses the request benefiting from the CoinGeckoProtocol’s parser, and acts for that request. You can see the mentioned loop in Figure 2. As the sockets are assigned a timeout, in case of a SocketTimeOutException (when waiting for a request), the object displays an informative message indicating that the client x

with IP/port is timed out, sends a timeout message to the client according to the protocol specifications and finally it terminates the connection by closing the socket. The most crucial function in the server class is “handleRequest(request)”. It takes the request from the user and takes action according to the “RequestType” field. You can see the actions taken in the Figure 3 (the code is very easy to read, so I shared it instead of explaining). Notice that in cases ”LIST” and “ID”, there is

```java
@Override
public void run(){
    CGPRequest request;
    String tempString;
    boolean control = true;
    try {
        while (control) {
            tempString = this.socketReader.readLine();
            request = CGPMessage.ParseRequest(tempString);
            control = this.handleRequest(request);
        }
    } catch (SocketTimeoutException e) {
        this.handleTimeoutSocket();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Figure 2

```java
switch (type) {
    case HEARTBEAT -> System.out.println("Heartbeat from: " + this.socket.getRemoteSocketAddress());
    case ACK -> System.out.println("Received unexpected ACK from: " + this.socket.getRemoteSocketAddress());
    case TERMINATE -> {
        System.out.println("Client is terminating!");
        this.disconnect();
        return false;
    }
    case LIST -> {
        try {
            JSONArray nftList = geckoAdapter.queryList();
            lastResponse = new CGPResponse(ResponseType.SUCCESS, BodyFormat.JSON_ARRAY, nftList.toString());
            this.socketWriter.println(lastResponse);
        } catch (GeckoAPIException apiException) {
            lastResponse = handleAPIException(apiException);
        }
        return this.waitForAck(lastResponse);
    }
    case ID -> {
        try {
            JSONObject nft = this.geckoAdapter.queryNFT(request.getBody());
            IDField[] idFields = IDField.getUniqueIds(request.getIdFields());
            String body = this.createIdResponseBody(nft, idFields);

            lastResponse = new CGPResponse(ResponseType.SUCCESS, BodyFormat.STRING, body);
            this.socketWriter.println(lastResponse);
        } catch (GeckoAPIException apiException) {
            lastResponse = handleAPIException(apiException);
        }
        return this.waitForAck(lastResponse);
    }
}
return true;
```

Figure 3

the call “waitForAck()”. This function waits for an acknowledgement message from the client during a predetermined time by sending the same message to the client periodically. Unless the “ACK” message is received, the server sends an “ACK_NOT_RECEIVED” message to the client and closes the connection. Errors stemming from the CoinGecko API are also part of the communication. If the CoinGecko API returns a message indicating that the user reached to the maximum number of retrievals, or a given NFT ID doesn’t exist, the server sends a “MAX_LIMIT_EXCEEDED” or “FAILURE” message together with an informative body.

Client Side

The Client module is composed of 2 classes. “GeckoClient” is responsible for taking input from the user, parsing and converting it to the format as specified by the protocol, and also displaying the response from the server in a readable format. On the other hand, “ConnectionGeckoServer” takes the user message as input, sends it to the server, waits for response and delivers it to the user again. It is also responsible for scheduling the sending of a “HEARTBEAT” message to the server (optional).

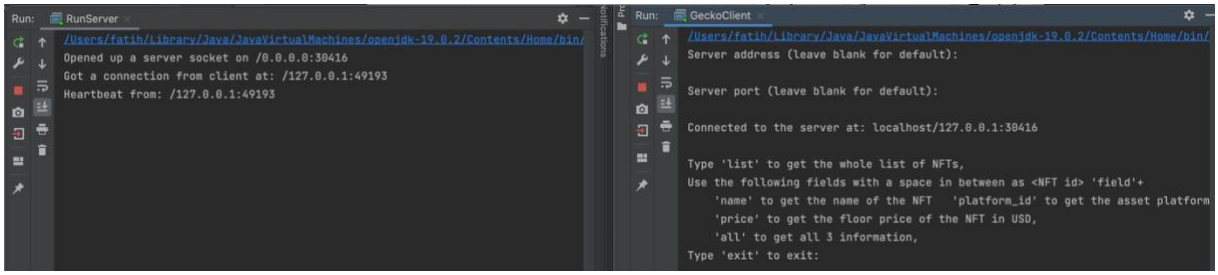
Here is how the client loop works: It displays an informative message giving instructions about how to provide an input and takes input from the user. Then it converts the string to a protocol message and passes to the “postRequest()” method of “ConnectionGeckoServer” object. That one, receiving the message, first checks whether there is a message in the input buffer of the socket. If there exists a message, it controls whether it is a “TIMEOUT” message or a “ACK_NOT_RECEIVED” message. If this is the case, it returns this message which will be processed by “GeckoClient”. If the message is another type of message, it simply ignores it because that means the server couldn’t receive the ACK message and resent the same message that was processes before. If this part doesn’t return the aforementioned messages from the server, the message generated by the client is sent to the server. Then “postRequest()” function waits for a response from the server. By receiving the message, it sends an “ACK” message to the server and returns the response to the client. Client, receiving the response, handles it according to the “Type” field. If the content of the response is needed to be displayed, then the “HandleResponse()” method displays it, but if the message indicates that the server is disconnected, then the “GeckoClient” closes the connection and ends the client loop. You can see the Figure 4 for some example case handlings.

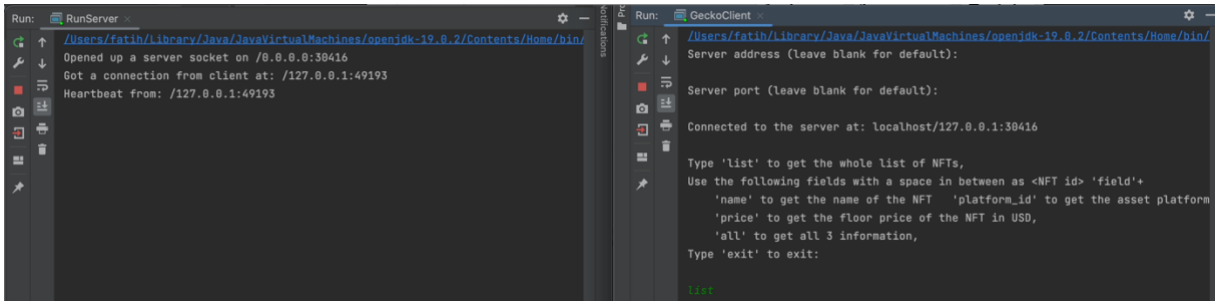
```
case ACK_NOT_RECEIVED:                                                    ⚠ 4
    System.out.println("Server didn't received ACK message! Disconnec
    connectionServer.disconnect();
    break;
case FAILURE:
    System.out.println("Failure! Message:");
    System.out.println(response.getBody() + "\n");
    break;
case SUCCESS:
    if (responseFormat == BodyFormat.STRING){
        System.out.println("Server response: ");
        System.out.println(response.getBody() + "\n");
    }
    else if (responseFormat == BodyFormat.JSON){
        JSONObject jsonObject = new JSONObject(response.getBody());
        String nftInfo = GetNftInfo(jsonObject);
        System.out.println(nftInfo);
    }
    else if (responseFormat == BodyFormat.JSON_ARRAY){
        JSONArray jsonArray = new JSONArray(response.getBody());
        System.out.println("Following NFTs are found:\n");
        for (int i = 0; i < jsonArray.length(); i++)
        {
            String nftInfo = GetNftInfo(jsonArray.getJSONObject(i));
            System.out.println(nftInfo + "\n");
        }
    }
    else{
        assert false: "Unsupported body format!";
    }
    break;
```

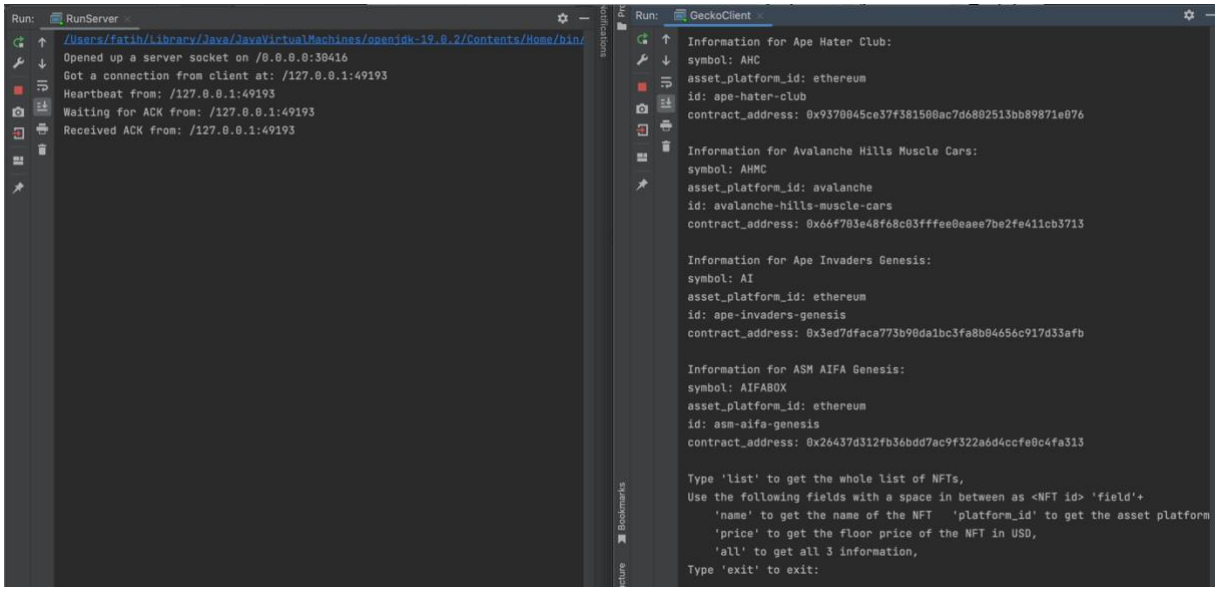
Figure 4

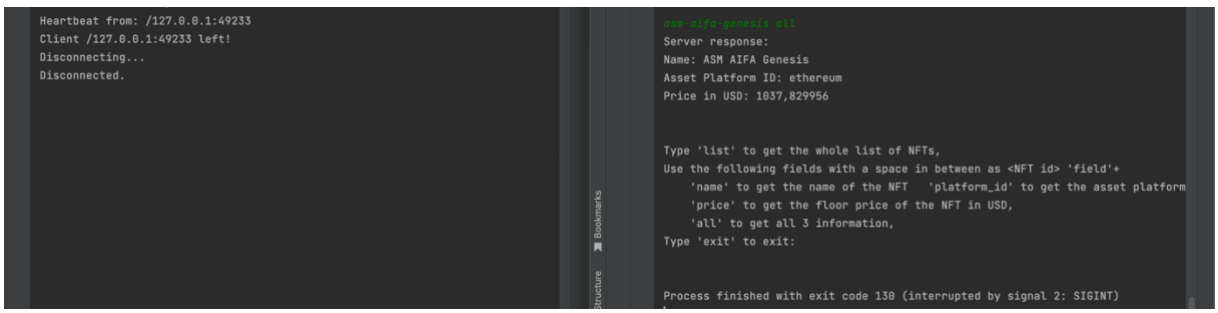
Test and Demos

Case 1: Server starts running. Client connects to the server (with heartbeat). Requests list of all NFT's. Requests price of a given NFT. Requests all fields (name, platform_id, price) of a given NFT. Client disconnects by stopping the program (A) or by typing exit (B).

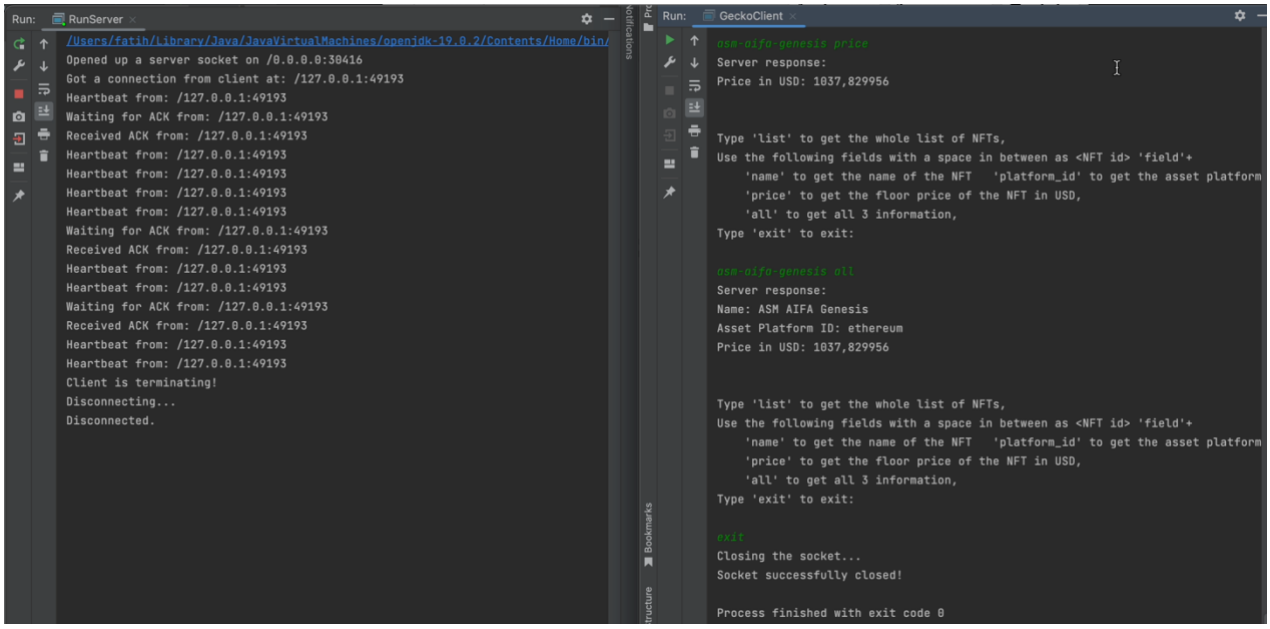








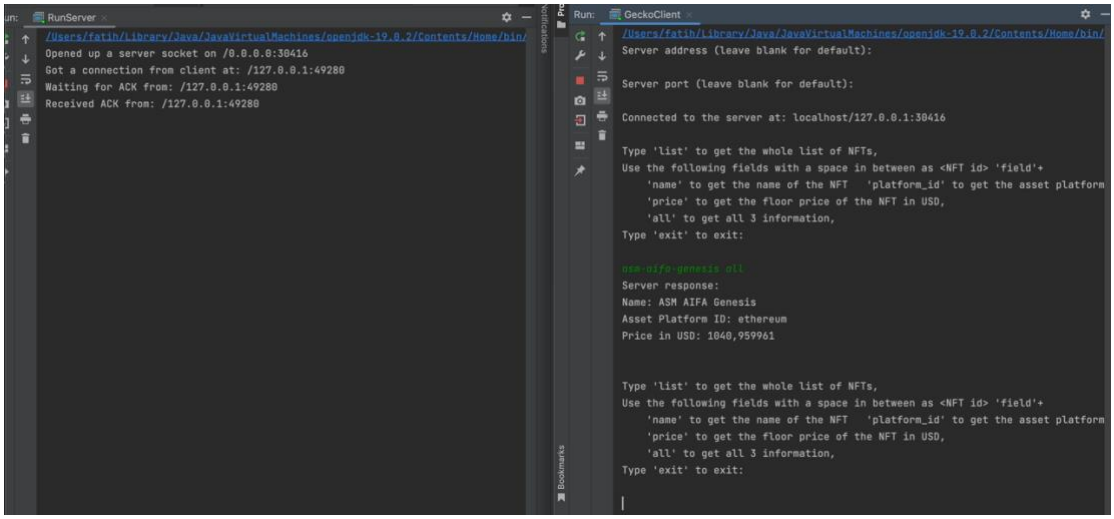
A

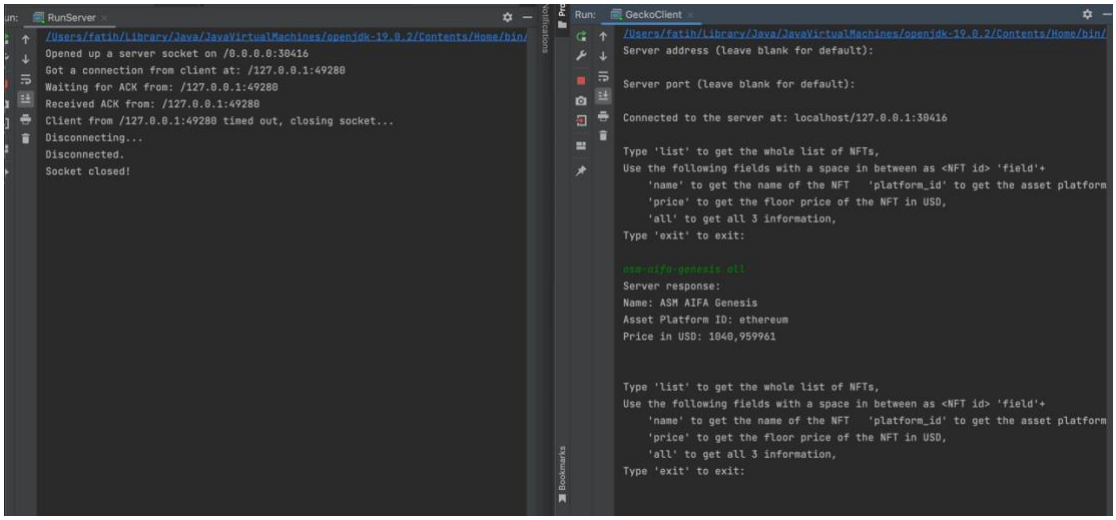


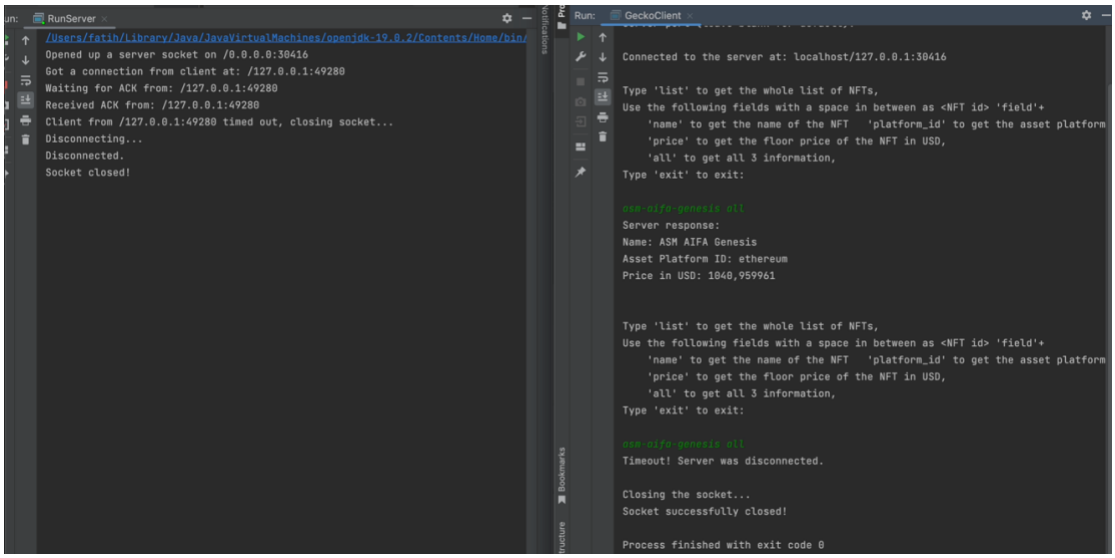
B

Notice that the informative message displayed on server side is differs between when the client exits by typing exit and by interrupting the program execution.

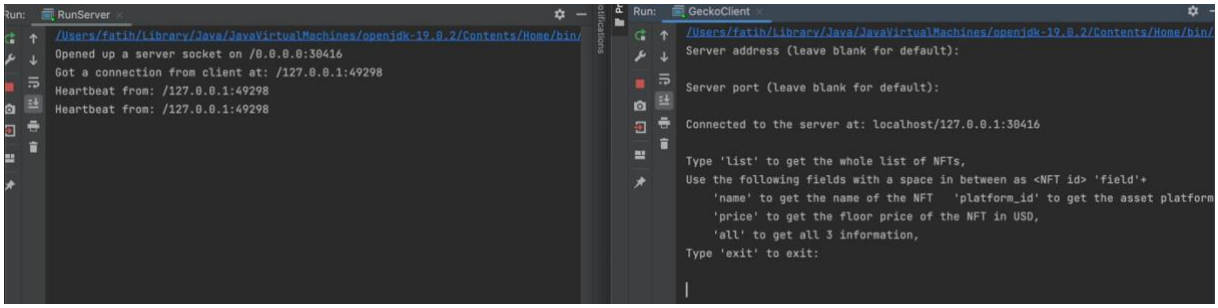
<u>Case 2</u>: Server starts running. Client connects without heartbeat. Client requests the price info of an NFT. Server closes the socket due to timeout. Client notices this when he/she tries to request something, and client closes the socket and program is terminated.

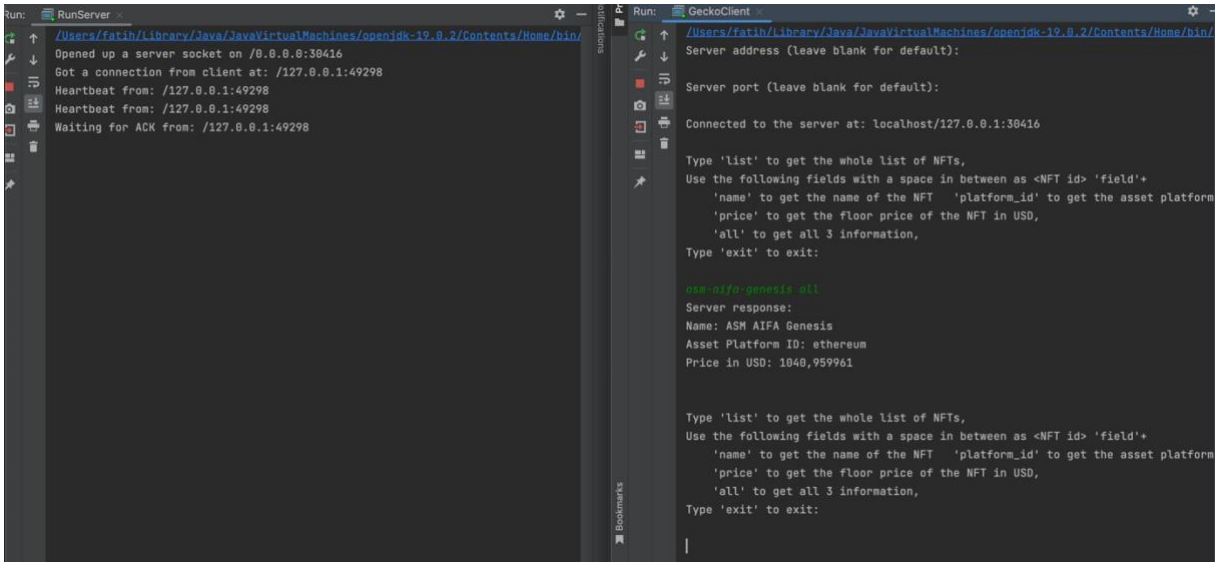


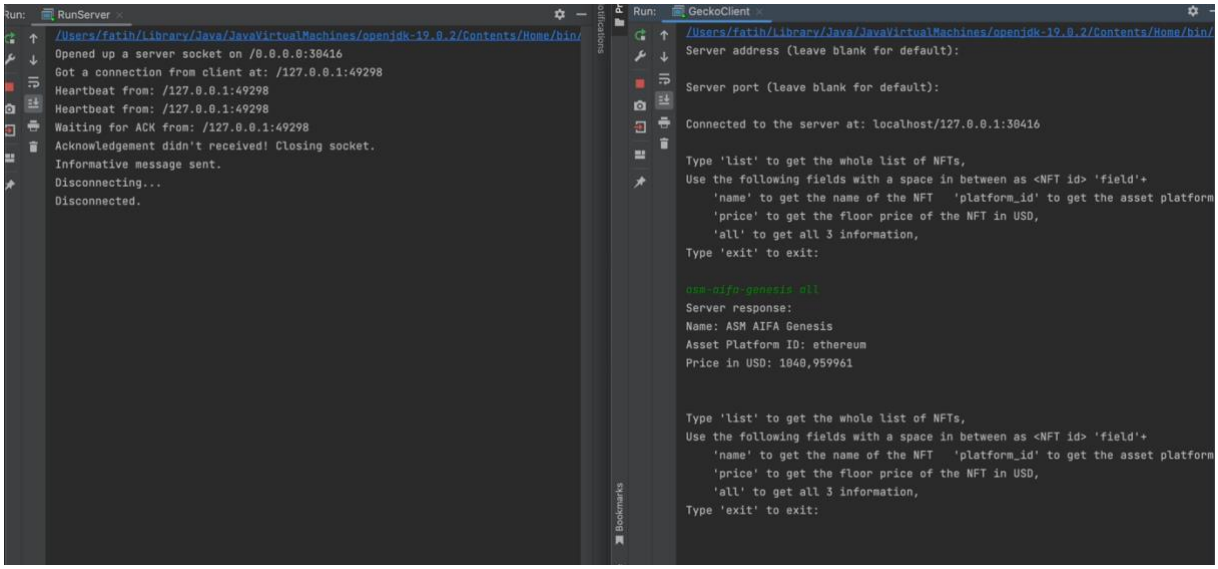


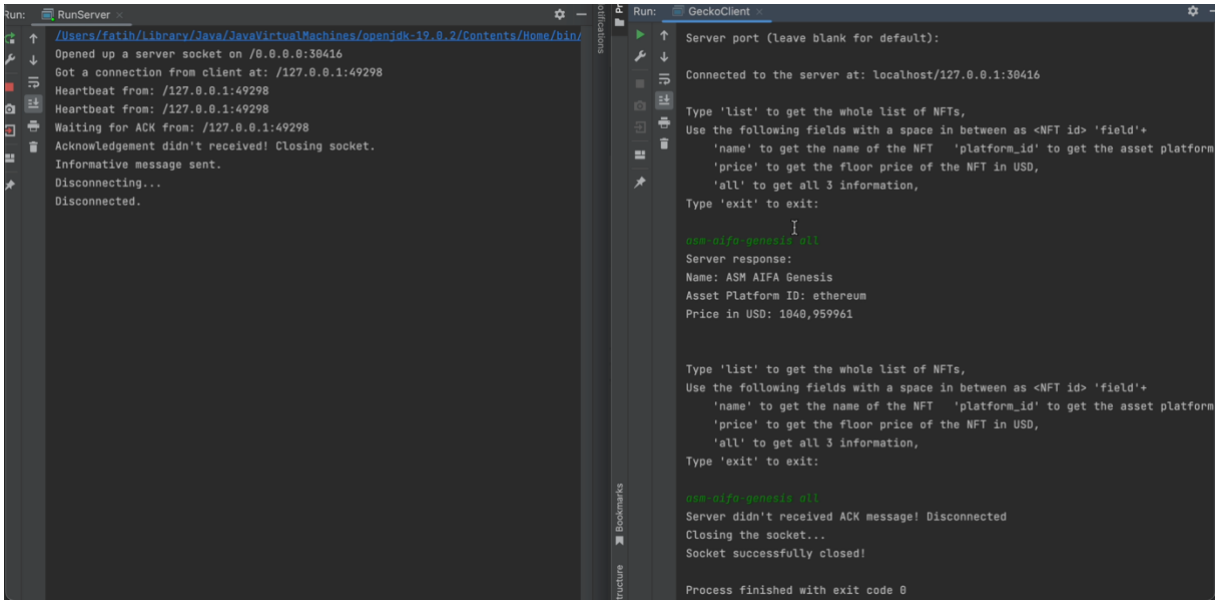


Case 3: Comment out the line that sends ACK to server. Run the server. Client connects with heartbeat. Request something. Server not receiving ack message disconnects. User realizes it after requesting another thing.

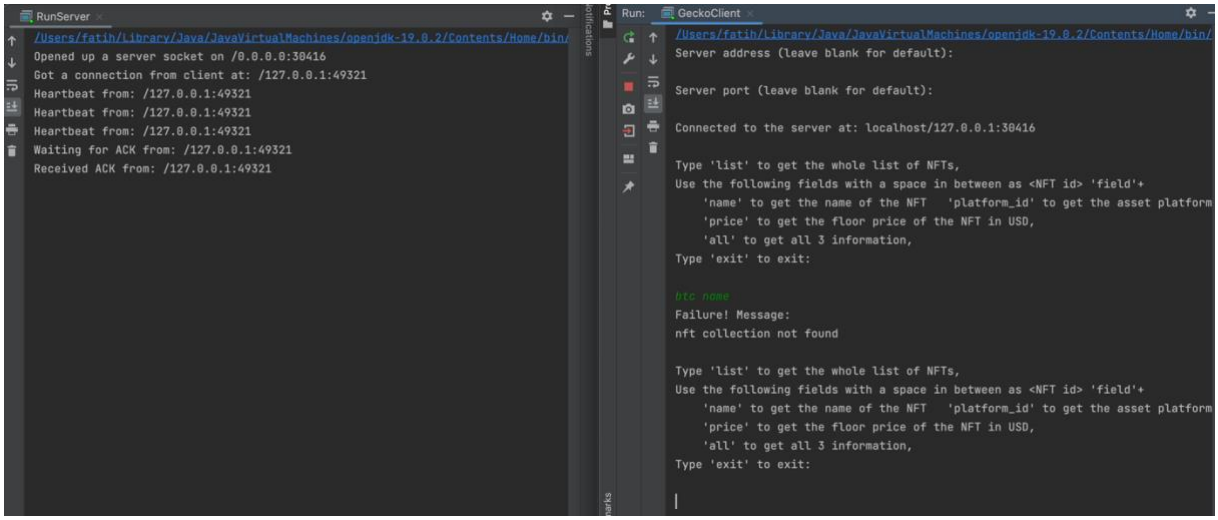


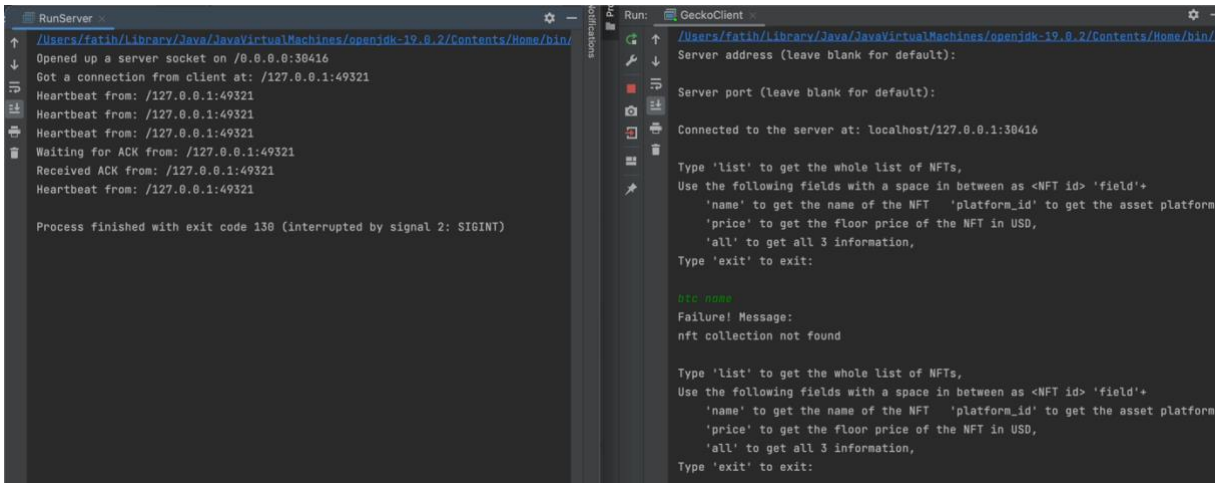


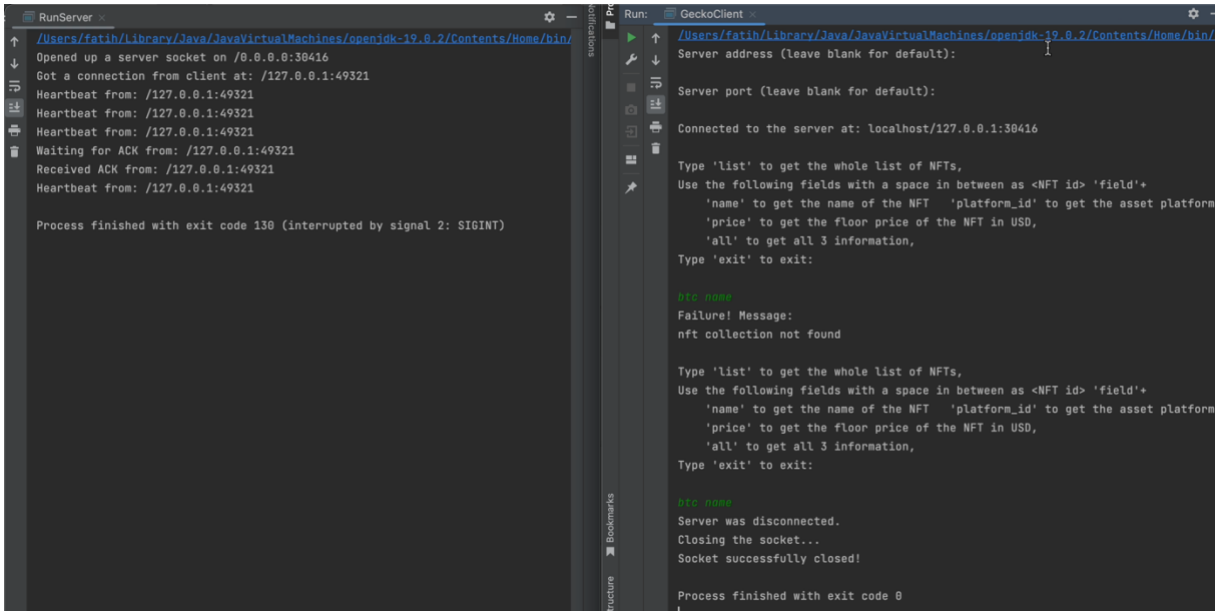



Run:
RunServer
/Users/fatih/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Home/bin/
Opened up a server socket on /0.0.0.0:30416
Got a connection from client at: /127.0.0.1:49298
Heartbeat from: /127.0.0.1:49298
Heartbeat from: /127.0.0.1:49298
Waiting for ACK from: /127.0.0.1:49298
Acknowledgement didn't received! Closing socket.
Informative message sent.
Disconnecting...
Disconnected.
Run:
GeckoClient
Server port (leave blank for default):
Connected to the server at: localhost/127.0.0.1:30416
Type 'list' to get the whole list of NFTs,
Use the following fields with a space in between as <NFT id> 'field'+
'name' to get the name of the NFT 'platform_id' to get the asset platform
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:
asm-aifa-genesis all
Server response:
Name: ASM AIFA Genesis
Asset Platform ID: ethereum
Price in USD: 1040,959961
Type 'list' to get the whole list of NFTs,
Use the following fields with a space in between as <NFT id> 'field'+
'name' to get the name of the NFT 'platform_id' to get the asset platform
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:
asm-aifa-genesis all
Server didn't received ACK message! Disconnected
Closing the socket...
Socket successfully closed!
Process finished with exit code 0
Bookmarks
Structure

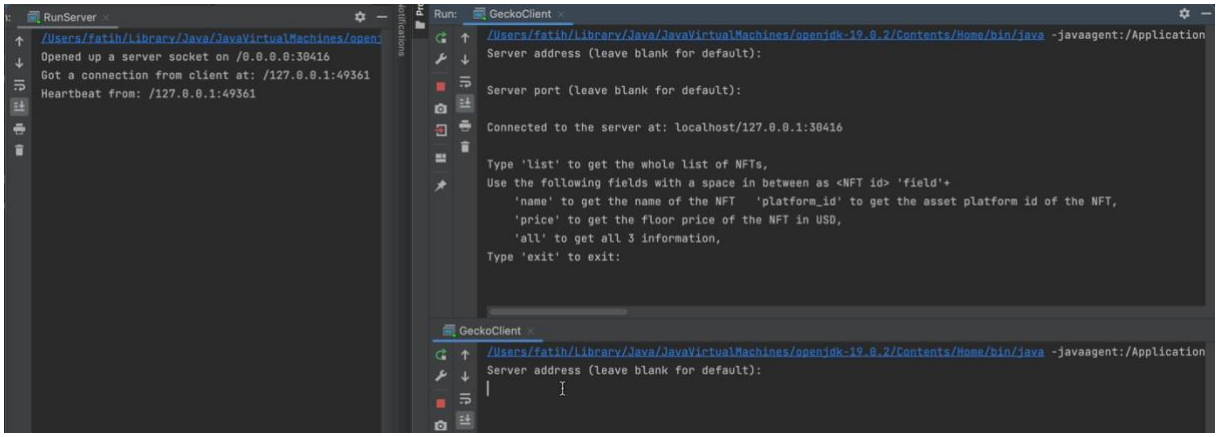
Case 4: Server start running. Client connects with heartbeat. Sends query for a non-existing NFT. Server message is displayed. Server is shut down. Client after trying to request something realizes this and closes the socket.

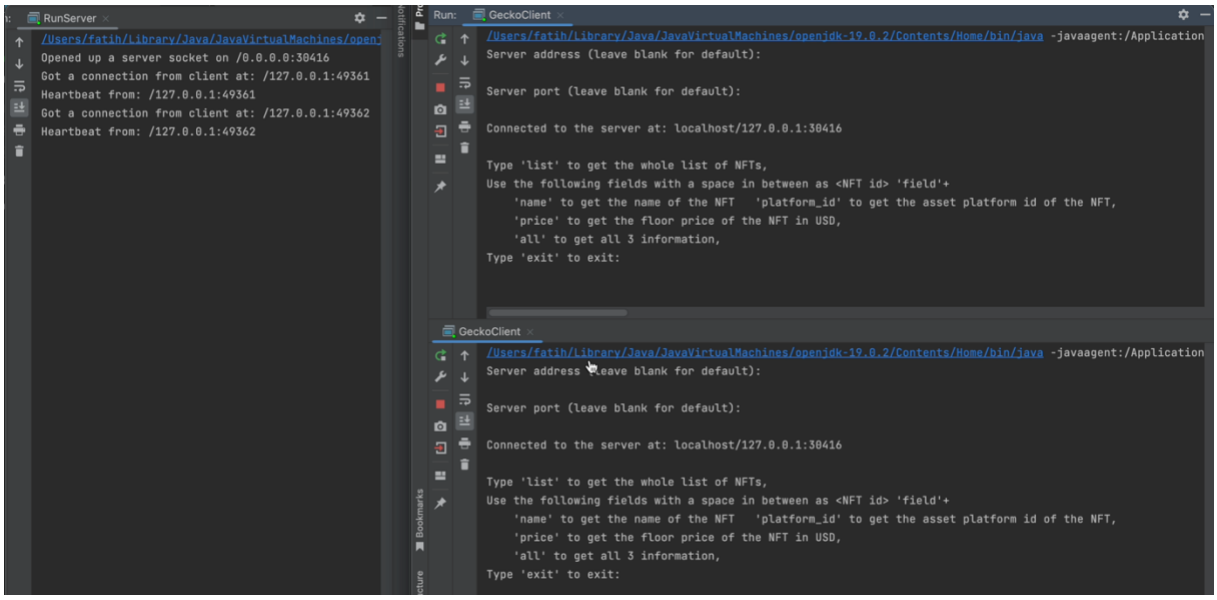


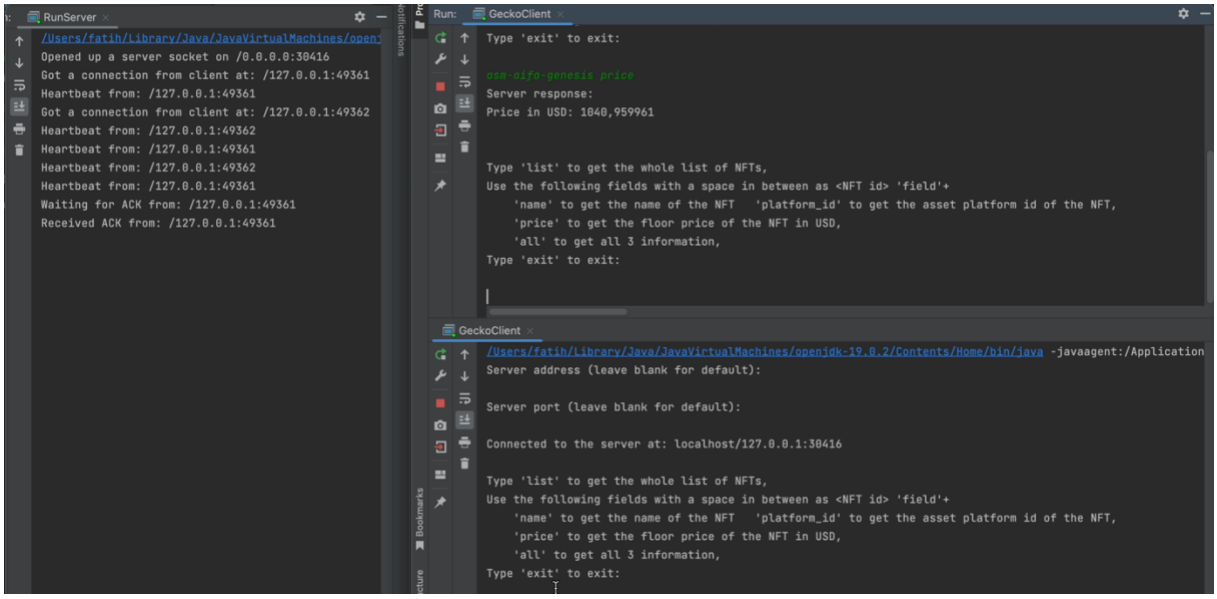


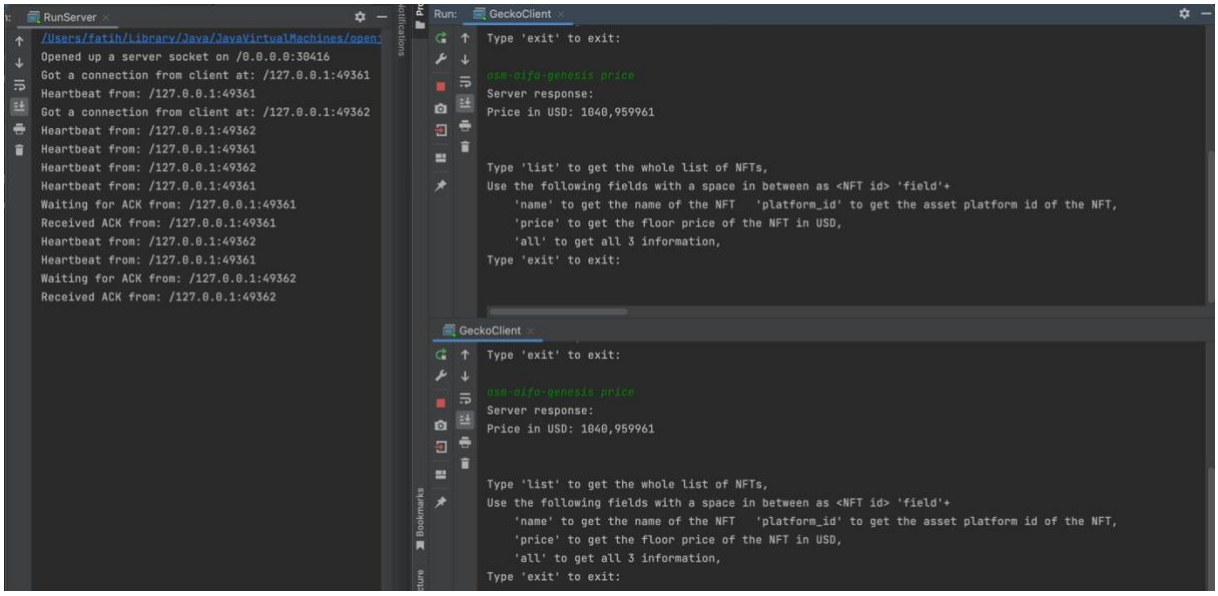


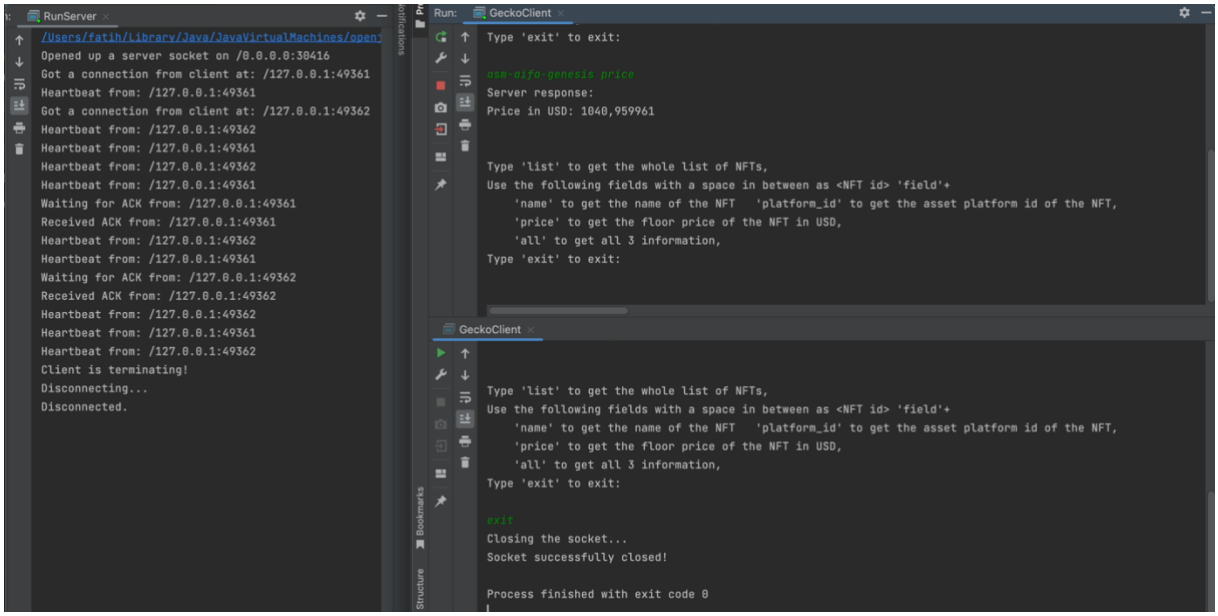
Case 5: 2 clients connect and makes some requests. Second client exits. First client requests too many things, exceeds the request limit and gets an informative message. First client interrupts the program.

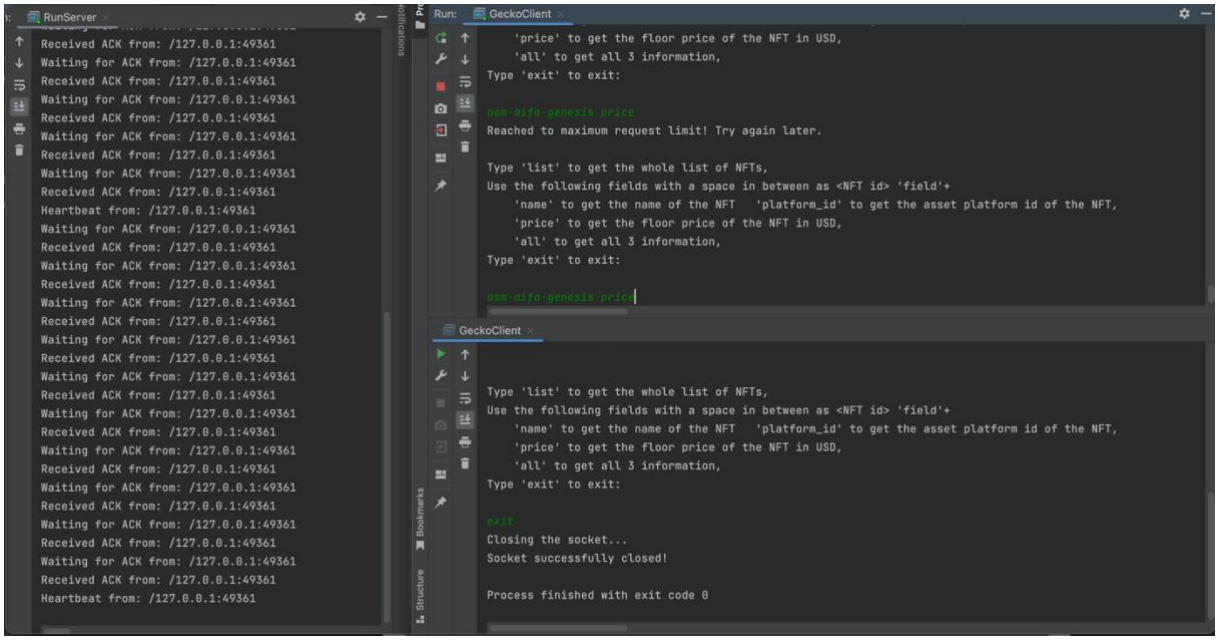


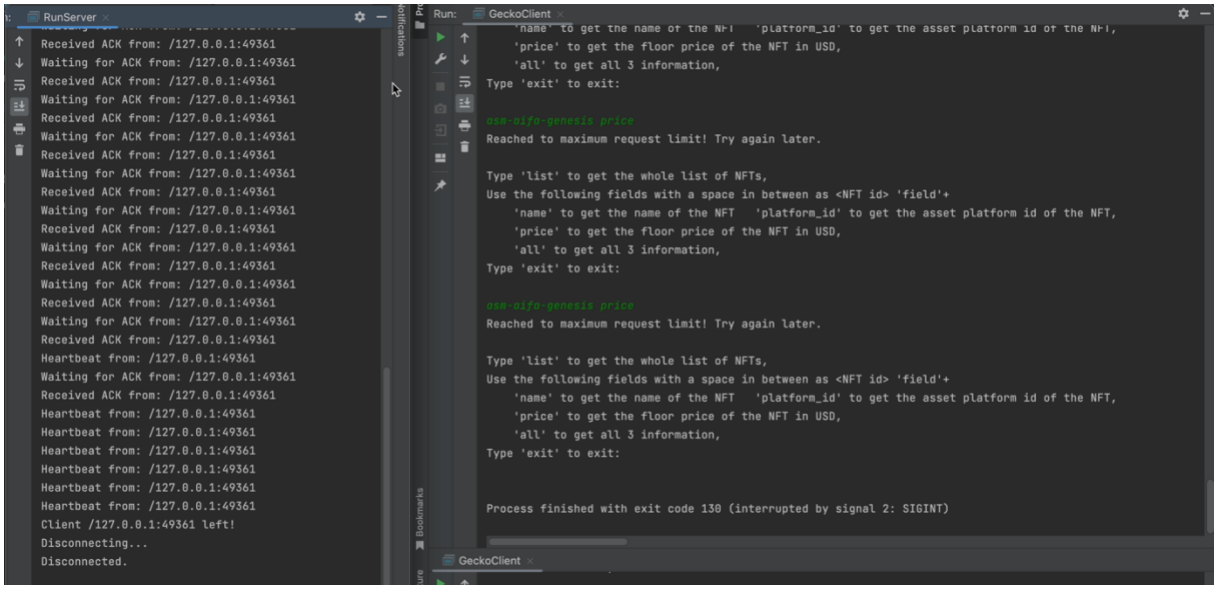





RunServer
/Users/fatih/Library/Java/JavaVirtualMachines/openj
Opened up a server socket on /0.0.0.0:30416
Got a connection from client at: /127.0.0.1:49361
Heartbeat from: /127.0.0.1:49361
Got a connection from client at: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49361
Heartbeat from: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49361
Waiting for ACK from: /127.0.0.1:49361
Received ACK from: /127.0.0.1:49361
Heartbeat from: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49361
Waiting for ACK from: /127.0.0.1:49362
Received ACK from: /127.0.0.1:49362
Run:
GeckoClient
Type 'exit' to exit:
asm-aifo-genesis price
Server response:
Price in USD: 1040,959961
Type 'list' to get the whole list of NFTs,
Use the following fields with a space in between as <NFT id> 'field'+
'name' to get the name of the NFT 'platform_id' to get the asset platform id of the NFT,
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:
GeckoClient
Type 'exit' to exit:
asm-aifo-genesis price
Server response:
Price in USD: 1040,959961
Type 'list' to get the whole list of NFTs,
Use the following fields with a space in between as <NFT id> 'field'+
'name' to get the name of the NFT 'platform_id' to get the asset platform id of the NFT,
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:


RunServer
/Users/fatih/Library/Java/JavaVirtualMachines/openj
Opened up a server socket on /0.0.0.0:30416
Got a connection from client at: /127.0.0.1:49361
Heartbeat from: /127.0.0.1:49361
Got a connection from client at: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49361
Heartbeat from: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49361
Waiting for ACK from: /127.0.0.1:49361
Received ACK from: /127.0.0.1:49361
Heartbeat from: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49361
Waiting for ACK from: /127.0.0.1:49362
Received ACK from: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49362
Heartbeat from: /127.0.0.1:49361
Heartbeat from: /127.0.0.1:49362
Client is terminating!
Disconnecting...
Disconnected.
Run:
GeckoClient
Type 'exit' to exit:
asm-aifo-genesis price
Server response:
Price in USD: 1040,959961
Type 'list' to get the whole list of NFTs,
Use the following fields with a space in between as <NFT id> 'field'+
'name' to get the name of the NFT 'platform_id' to get the asset platform id of the NFT,
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:
GeckoClient
Type 'list' to get the whole list of NFTs,
Use the following fields with a space in between as <NFT id> 'field'+
'name' to get the name of the NFT 'platform_id' to get the asset platform id of the NFT,
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:
exit
Closing the socket...
Socket successfully closed!
Process finished with exit code 0


RunServer
Received ACK from: /127.0.0.1:49361
Waiting for ACK from: /127.0.0.1:49361
Heartbeat from: /127.0.0.1:49361
Run:
GeckoClient
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:
asm-aifa-genesis price
Reached to maximum request limit! Try again later.
Type 'list' to get the whole list of NFTs,
Use the following fields with a space in between as <NFT id> 'field'+
'name' to get the name of the NFT 'platform_id' to get the asset platform id of the NFT,
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:
exit
Closing the socket...
Socket successfully closed!
Process finished with exit code 0


RunServer
Received ACK from: /127.0.0.1:49361
Waiting for ACK from: /127.0.0.1:49361
Heartbeat from: /127.0.0.1:49361
Run:
GeckoClient
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:
asm-aifa-genesis price
Reached to maximum request limit! Try again later.
Type 'list' to get the whole list of NFTs,
Use the following fields with a space in between as <NFT id> 'field'+
'name' to get the name of the NFT 'platform_id' to get the asset platform id of the NFT,
'price' to get the floor price of the NFT in USD,
'all' to get all 3 information,
Type 'exit' to exit:
Process finished with exit code 130 (interrupted by signal 2: SIGINT)
Client /127.0.0.1:49361 left!
Disconnecting...
Disconnected.