COMP416: Computer Networks Project 1

NFTNet: CoinGecko-API Based Application Layer Protocol

Emir Fatih AYYILDIZ 71527

This project work is about the application layer of the network protocol stack. It involves application layer protocol principles, design and implementation of a client/server protocol, application layer software development, socket programming, and multithreading.
This project is an application layer protocol with the application programming interface (API) of CoinGecko. There are 2 APIs for Non-Fungible Token (NFT) information extraction from the CoinGecko server. First one is used for listing all NFT data on the server. Second one is used for getting name, id and floor price of the specified NFT.

Protocol Overview:

I have used TCP connection to communicate between client and server. The TCP connection ensures reliable, ordered, and error-checked delivery of data between the client and server, making it suitable for scenarios where data integrity is crucial.

1- Request Types:

List NFTs:
Request: ""
Response: A JSON array containing information about all NFTs.
Client Output: A string containing name and IDs of all NFTs.

Search for an NFT:
Request: NFT ID
Response: A JSON object containing information about the specific NFT.
Client Output: Name, platform id and floor price of the specific NFT

2- Error Handling:

In case of errors, the server can respond with an error message specifying the issue.

3- Timeout Handling:

If the client doesn't respond within a specified timeout, the server can handle the timeout exception and close the client socket.

Usage:

1- Server code starts. Server shows the Inet4 address and waits for a client to connect. When a client connects, a thread starts and prints the socket address information. The server works simultaneously for multiple clients.
2- Client sends the request. The request can be empty, NFT id or QUIT.
3- The server gets the client message and gets data from the CoinGecko's API. Data is parsed and the result is sent to the client.
4- If a new message is sent from the client, server keeps the connection. Otherwise, the server closes the socket for that client in 15 seconds.

Overview:

The server code starts with the default server code 4444 on multithreadserver file. A socket is created, and the address is printed. Then, the code works on a while loop to start the function called listenandaccept. The function creates a socket for the client and creates a thread for them. This allows server to accept many clients and serve them at the same time.

```
private void listenAndAccept()
{
    Socket s;
    try
    {
        s = serverSocket.accept();
        System.out.println("A connection was established with a client on the address of "
        ServerThread st = new ServerThread(s);
        st.start();
    }
```

The thread starts the serverthread file. Communication between the client and server is done by using input and output streams. When the client connects to the server, you can type QUIT to end the connection. Sending an empty message will ask for the server a list of all NFTs names and ids. Typing the id of an nft will ask the server for name, platform id and floor price of the specified nft. When receiving an empty message, which has no parameter, the server will connect to coingeckos API and get data. data will be converted to JSON array and parse the name and id of every NFT.

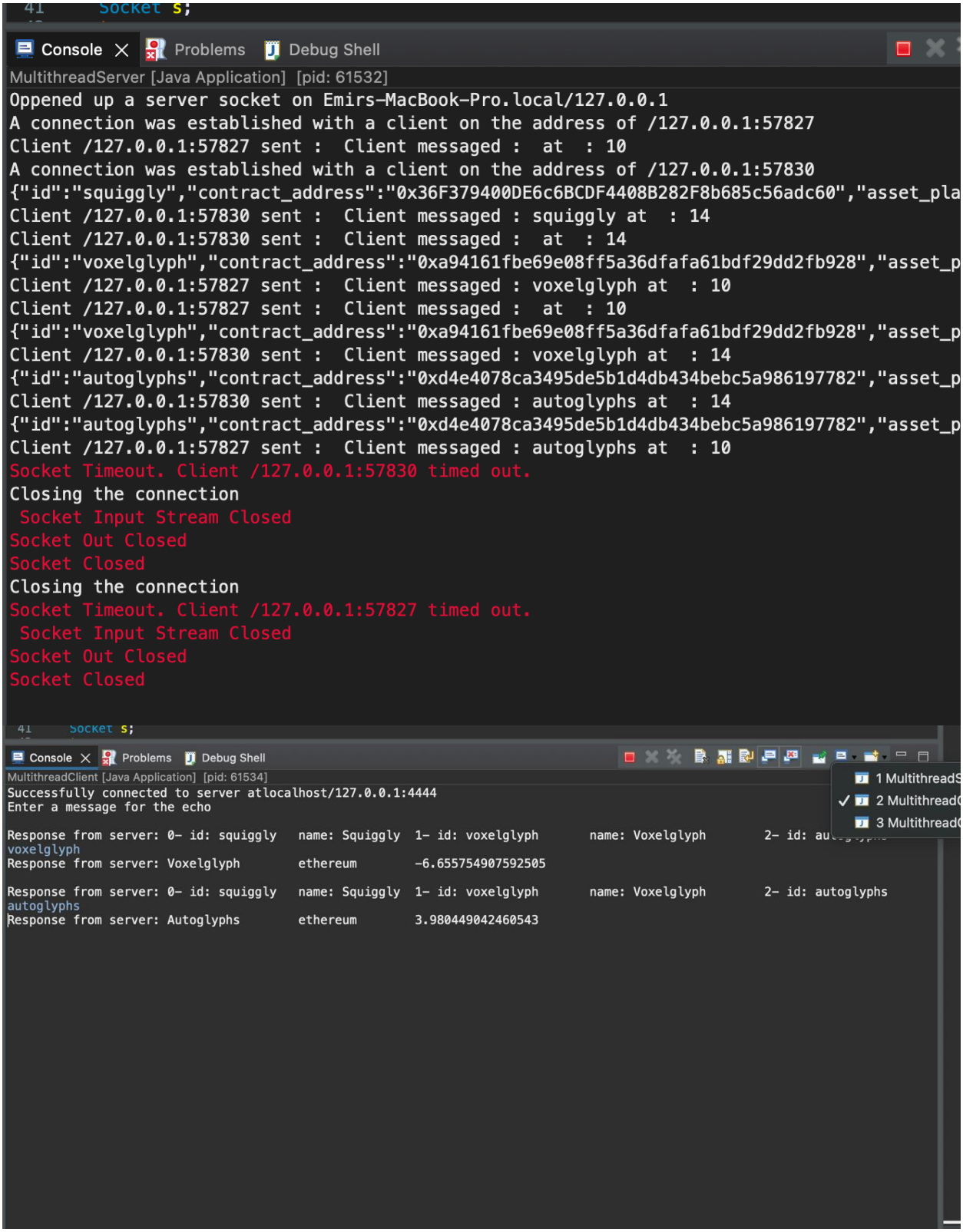
```
String inline = "";
Scanner scanner = new Scanner(url.openStream());
while (scanner.hasNext()) {
    inline += scanner.nextLine();
}
//Close the scanner
scanner.close();
JSONArray jsonObject = new JSONArray(inline);
for (int i = 0; i < jsonObject.length(); i++)
{
    String id = jsonObject.getJSONObject(i).getString("id");
    String name = jsonObject.getJSONObject(i).getString("name");
    info += i + "- " + "id: " + id + "\t" + "name: " + name + "\t";
    //System.out.println(info);
}
return info;
```

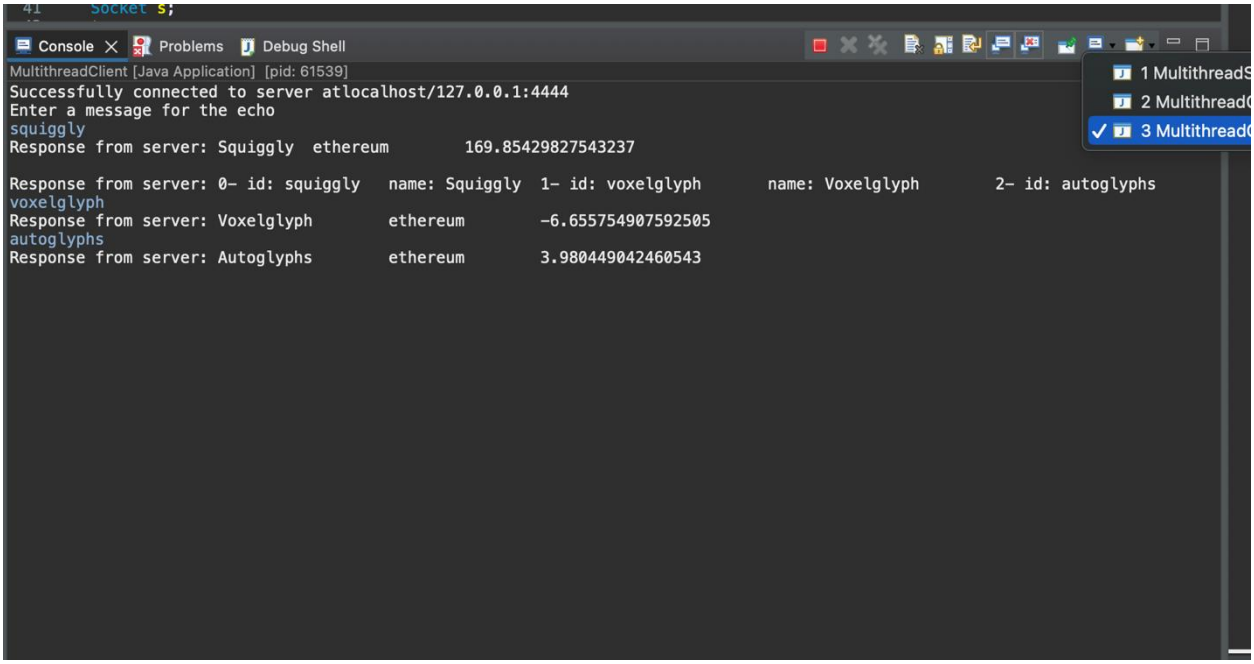
Names and ids will be stored in a string and will be sent to the client. Client prints the string to see all NFTs. Thread id and message of the client can be seen on the server side, as well as their socket addresses.

When the client sends the id of an NFT, server makes a request to the API about the specified NFT. This data has more information about the NFT. We parse the name, platform id and floor price of the NFT and store them in a string variable. This string is sent to the client and client prints the data

```
//System.out.println(inline);

System.out.println(inline);
String makeJson = "[" + inline + "]";
JSONArray jsonObject = new JSONArray(makeJson);
String name = jsonObject.getJSONObject(0).getString("name");
String asset_platform_id = jsonObject.getJSONObject(0).getString("asset_platform_id");



JSONArray jsonArray = new JSONArray(makeJson);
JSONObject myjsonObject = jsonArray.getJSONObject(0);
double floorPrice7dPercentageChange = myjsonObject.getJSONObject("floor_price_7d_percentage_

info = name + "\t" + asset_platform_id + "\t" + floorPrice7dPercentageChange;
```

Here, an example is shown from the server side. You can see that upon starting the server, the server details are given. When a client connects, its details are printed. When a client sends a message, their message is shown as well as their thread id. When these clients stay idle for 15 seconds, the server closes the socket for that client.


41 Socket s;
Console
Problems
Debug Shell
MultithreadServer [Java Application] [pid: 61532]
Oppened up a server socket on Emirs-MacBook-Pro.local/127.0.0.1
A connection was established with a client on the address of /127.0.0.1:57827
Client /127.0.0.1:57827 sent : Client messaged : at : 10
A connection was established with a client on the address of /127.0.0.1:57830
{"id":"squiggly","contract_address":"0x36F379400DE6c6BCDF4408B282F8b685c56adc60","asset_pla
Client /127.0.0.1:57830 sent : Client messaged : squiggly at : 14
Client /127.0.0.1:57830 sent : Client messaged : at : 14
{"id":"voxelglyph","contract_address":"0xa94161fbe69e08ff5a36dfafa61bdf29dd2fb928","asset_p
Client /127.0.0.1:57827 sent : Client messaged : voxelglyph at : 10
Client /127.0.0.1:57827 sent : Client messaged : at : 10
{"id":"voxelglyph","contract_address":"0xa94161fbe69e08ff5a36dfafa61bdf29dd2fb928","asset_p
Client /127.0.0.1:57830 sent : Client messaged : voxelglyph at : 14
{"id":"autoglyphs","contract_address":"0xd4e4078ca3495de5b1d4db434bebc5a986197782","asset_p
Client /127.0.0.1:57830 sent : Client messaged : autoglyphs at : 14
{"id":"autoglyphs","contract_address":"0xd4e4078ca3495de5b1d4db434bebc5a986197782","asset_p
Client /127.0.0.1:57827 sent : Client messaged : autoglyphs at : 10
Socket Timeout. Client /127.0.0.1:57830 timed out.
Closing the connection
Socket Input Stream Closed
Socket Out Closed
Socket Closed
Closing the connection
Socket Timeout. Client /127.0.0.1:57827 timed out.
Socket Input Stream Closed
Socket Out Closed
Socket Closed
41 Socket s;
Console
Problems
Debug Shell
MultithreadClient [Java Application] [pid: 61534]
Successfully connected to server atlocalhost/127.0.0.1:4444
Enter a message for the echo
Response from server: 0- id: squiggly name: Squiggly 1- id: voxelglyph name: Voxelglyph 2- id: au
voxelglyph
Response from server: Voxelglyph ethereum -6.655754907592505
Response from server: 0- id: squiggly name: Squiggly 1- id: voxelglyph name: Voxelglyph 2- id: autoglyphs
autoglyphs
Response from server: Autoglyphs ethereum 3.980449042460543
1 MultithreadS
2 MultithreadC
3 MultithreadC


41 Socket s;
Console
Problems
Debug Shell
MultithreadClient [Java Application] [pid: 61539]
Successfully connected to server atlocalhost/127.0.0.1:4444
Enter a message for the echo
squiggly
Response from server: Squiggly ethereum 169.85429827543237
Response from server: 0- id: squiggly name: Squiggly 1- id: voxelglyph name: Voxelglyph 2- id: autoglyphs
voxelglyph
Response from server: Voxelglyph ethereum -6.655754907592505
autoglyphs
Response from server: Autoglyphs ethereum 3.980449042460543
1 MultithreadS
2 MultithreadC
3 MultithreadC