KOÇ ÜNİVERSİTESİ
COMPUTER ENGINEERING

COMP416: Computer Networks

Project 1

NFTNet: CoinGecko-API Based Application Layer Protocol

Due: 09/11/2023, 11:59 pm (Late submissions will not be accepted).

Submission of the project deliverables is via Blackboard.

This is an individual project. You are not allowed to share your codes with each other.

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.

Through this project, you are going to develop an application layer protocol by interacting with the application programming interface (API) of CoinGecko. You are required to work with the following APIs for Non-Fungible Token (NFT) information extraction from the CoinGecko server:

1. List NFT data API
2. Search for an NFT API

These are two separate APIs that allow users to interact with the CoinGecko aggregated data and retrieve the required information related to NFTs.

Project Overview

In this project, you are asked to develop a CoinGecko API-based application layer protocol based on the client/server model. NFTNet server uses a TCP connection to interact with the clients. Fig.1 shows connections for a sample NFTNet server and client interactions. As shown in the figure, only the NFTNet server interacts with the CoinGecko API web server and provides the NFTNet client with the information required.

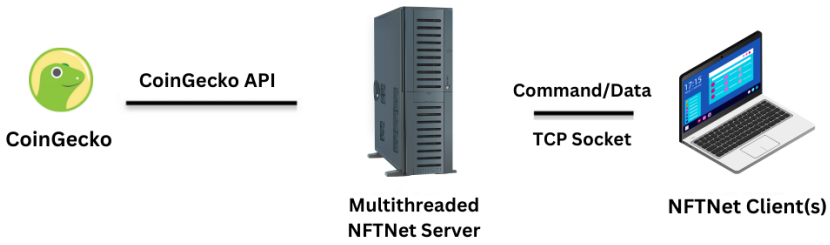

Figure 1. The NFTNet Client/Server connections.

Implementation Details

The NFTNet application layer protocol has four main components:

- Interaction of the CoinGecko API and the server
- Server side of the application
- Client side of the application
- Client-Server Communication Protocol

It is pertinent to note that the mentioned APIs allow 50 calls/minute. That means a maximum of 50 API calls (method invocation) can be done each minute. The API will give a different response (error message) when trying to make more than 50 calls in a minute. The developed application must keep these limits in perspective.

Methods

1. List NFT data API can be called with no parameters and returns a JSON array containing information about all the NFTs. Each NFT is represented as a JSON object with id, contract address, name, asset platform id, and symbol fields. The endpoint for this API is https://api.coingecko.com/api/v3/nfts/list.
2. Search for an NFT data API (https://api.coingecko.com/api/v3/nfts/{id}). This API requires the id of the NFT that we need to get its data.

NFTNet Sequence overview:

1. NFTNet Client/s connect with the server and display the connected socket information.
2. Client/s send the query to the server formatted as per the protocol specifications.

3. The NFTNet server upon receiving the query, extracts the relevant part and formulates the request to be sent to the API server.
4. The NFTNet server establishes a connection with the required CoinGecko API and fetches the required information (JSON).
5. The NFTNet server communicates the results to the concerned client per the communication protocol.
6. The client acknowledges receiving the information as per the communication protocol and displays the received information to the user.
7. Clients may then indicate their willingness to send additional queries or terminate the connection.
8. The server may timeout the socket of a client if it does not receive any further messages from the client/s.

NFTNet Server side overview:

The server for the NFTNet application should:

1. Establish a connection with the respective CoinGecko web servers using the APIs and download the specified required information.
   (a) Ask for a list of NFTs
   (b) Ask for the name, asset_platform_id, and price in USD for an NFT.
2. Allow multiple clients to connect using multi-threading. Each thread is responsible for serving a client.
3. Attach a timeout with the client socket (not the welcoming socket).

Based on the client's request, the server will parse the client input and use the API to extract the relevant information from the CoinGecko server. This information will then be passed to the client.

The process on the server side would be:

1. Create a welcoming socket
2. Accept incoming client connections at the welcoming socket, and assign a thread for each connected client.
3. Parse the requests coming in from the clients and query the required information from the web server. (The protocol for forwarding requests between the client and the server has to be designed and developed by yourself and explained in your report.)
4. Send the requisite information to the client complying with the designed protocol.

5. Terminate connection on a timeout or upon receiving a terminate "type" message from the client.

NFTNet Client Side Overview:

This application envisions multiple clients interacting with a single server. For this application, the clients should:

1. Be able to submit requests to the server using the designed protocol.
2. Be able to receive data from the server and display it to the user.

Client-Server Interaction

The client-side must take care of the parameters to be sent to the server. The process on the client side will follow the given steps:

1. Initiate connection with the server
2. Pass the requests to the server using the designed protocol.
3. Receive the information sent over by the server, and parse it depending on the protocol.
4. Display the information in the appropriate manner.
5. Terminate the connection in case appropriate data is received and no other request is forwarded within the timeout duration. (Appropriate tests for demonstration purposes should be developed.)

Protocol Design

All communications between the client and server must follow a set protocol. A communication protocol is a system of rules that allows two or more entities of a communications system to exchange information. The communication protocol defines the format for exchanging various types of messages. The entire payload is divided into fields, each of which represents a specific purpose.

This project envisions you designing your own communication protocol for message exchange between the client and the server. All the communication between the client and the server after the initial handshake and assigning of the connected port must happen according to this protocol. The protocol you are going to design should be inspired by the HTTP protocol but with your own codes and fields.

You are required to envision the interaction between the client and the server and create a complete protocol for this communication. Each client and server should have access to a range of communication messages which may be chosen by specifying a particular value of a field. The client and the server will handle the received messages according to the protocol you designed. Similarly, they will construct messages adhering to the protocol.

The protocol must be clearly defined in the code as well as the project report. The documentation of the protocol in your report should allow anyone who reads it to write programs that adhere to your protocol.

Initialization Phase

At the start, the server allows connection for multiple clients. After the connection, the client/s sends a query message to the server to obtain the required information. The server queries the CoinGecko server using the API for the required information. Once received, the server reformats the received information according to the communication protocol and sends it back to the client.

Here are some useful links:

- Data Input Stream
- Data Output Stream
- Filter Output Stream
- Java - Multithreading
- JSON-java

Execution Scenario

The client and server are expected to reside on a single machine for simplicity of both execution and demonstration. The project envisions you connecting multiple clients and one server.

Project deliverables

You should submit your source code and project report (in a single .rar or .zip file) via Blackboard.

Use the following naming for the file: "Project1-<your-name-surname-KUSIS-ID>"

Report

Report: The report should start with a very brief description of the project. This should be followed by an explanation of the philosophies, especially the **client-server application layer protocol design and specifications**. Following this, you should provide an overview of the programming of the client and the server side including connection with the API and data transmitted back to the client. Instead of attaching complete code, it is strongly recommended to attach code Snippets and images of the results of their execution. **The report should**

explicitly describe the test scenarios developed to evaluate the full range of features required for the NFTNet.

Source Code

A .zip or .rar file that contains your implementation in two separate projects (one for the client and another for the server) using Eclipse, NetBeans, or IntelliJ IDEA. If you aim to implement your project in any IDE other than the mentioned ones, you should first consult with the TAs and get confirmation.

Demonstration

You are required to demonstrate the execution of the NFTNet Application for the defined requirements. Your demo sessions will be announced by the TAs. Attending the demo session is required for your project to be graded. The on-time attendance at the demo session is considered a grading criterion. During your demonstration, you will be asked to demonstrate at least two clients being connected to the server at the same time. During this time, you will be asked to display all the operations of the application through a series of questions. Please prepare the codes in an appropriate manner to answer the range of features as defined for the NFTNet application. You have the creative freedom to present any additional features you may have built into the application in any way you deem feasible.

Good Luck!