NFTNet Project

COMP416 – Computer Networks Project 1

Ahmet Uyar

1. Project Description:

The NFTNet project aims to create a client-server application for retrieving information about Non-Fungible Tokens (NFTs) from the CoinGecko API. The client allows users to list NFTs and get details about a specific NFT using a client-server communication protocol.

2. Philosophies and Design:

The project follows a client-server application layer protocol for communication. The protocol involves two main request types: "LIST" to get a list of NFTs and "SEARCH" to retrieve details about a specific NFT by ID. The server connects to the CoinGecko API to fetch the required information to later send the information to the client in line with the communication protocol. Then, the information is parsed and displayed to the user in a request-specific manner.

3. Client-Side Programming:

The client-side of NFTNet is implemented in Java, featuring a straightforward command-line interface to enhance user interaction. This section provides an in-depth look into the design and implementation, illustrating how the client establishes a connection with the server, sends requests, and processes server responses.

Client Initialization: The client application begins by initializing a socket connection with the server. The Socket class in Java facilitates this connection, allowing communication over the established socket.

User Interaction: The client provides a user-friendly command-line interface, presenting a menu that allows users to choose between different actions: listing NFTs, getting details about a specific NFT, or exiting the application.

Sending Requests: The client sends requests to the server based on user input. The PrintWriter stream is used to send these requests. In this example, the client requests a list of NFTs.

Processing Server Responses: The client-side involves processing responses received from the server. A separate thread is dedicated to handling server responses to prevent blocking the main thread.

The client-side programming for NFTNet involves not just establishing a connection with the server but also creating an intuitive user interface. Employing a menu-driven approach allows users to easily navigate and interact with the application. This section has outlined essential aspects of the client implementation, covering the initialization of the connection and the processing of server responses. The following section will offer an in-depth examination of the server-side programming and its integration with the CoinGecko API.

4. Server-Side Programming:

The server is also implemented in Java and uses multi-threading to handle multiple client connections simultaneously. It communicates with the CoinGecko API to fetch NFT information based on client requests. The server validates and processes incoming requests, providing appropriate responses.

Efficient Client Communication: The client-side programming of NFTNet establishes a connection with the server and provides an intuitive user interface with a menu-driven approach, enhancing user interaction and navigation.

Key Aspects of Client Implementation: This section highlights critical elements of the client implementation, ranging from initializing connections to processing server responses based on user input.

Multi-threaded Server Design: The server-side implementation utilizes a multi-threaded design in Java, ensuring concurrent connections are efficiently handled for streamlined communication and effective data retrieval.

Request Decoding and Verification: Upon receiving a client request, the server decodes the message, identifying the requested operation type and verifying the format to conform to the predefined protocol.

Handling "LIST" Requests: For "LIST" requests, the server communicates with the CoinGecko API, retrieves a list of available NFTs, formats the data into a response message, and transmits it to the client for display.

Handling "SEARCH" Requests: For "SEARCH" requests, the server extracts the NFT ID, queries the CoinGecko API for detailed information, and formats the retrieved data into a response message sent to the client.

5. Connection with CoinGecko API:

The server connects to the CoinGecko API using HTTP requests. It fetches NFT list data and specific NFT details based on the client's request. The server parses the API responses and sends relevant information back to the client.

6. Communication Protocol:

Message Format: Messages maintain a well-defined structure with sections separated by specific delimiters. A typical message includes the request type, parameters, and payload, all organized systematically.

Request-Response: Following a request-response model, the client initiates communication with requests like "LIST" and "SEARCH." The server processes these requests, generating responses containing "RES" (indicating a response) and "SUCCESS" to denote a successful operation. This paradigm ensures coherent interaction.

Message Parsing: Incoming messages on the client side are parsed to extract essential information. Parsing involves breaking down the message into sections using predefined delimiters, allowing the client to discern request type and associated parameters. Similarly, the server parses incoming requests to interpret user intent and execute actions.

Error Handling: Robust error-handling mechanisms are integral to the protocol, ensuring graceful recovery from unexpected scenarios. Error messages follow a standardized format, featuring "RES" and "ERROR" keywords. This approach enhances reliability and fault tolerance.

User Input Processing: The client collects user input via a command-line interface, translating it into requests compatible with the protocol. The processing mechanism validates user commands, minimizing input-related errors. Responses include "RES" and "SUCCESS" for effective communication.

Request Handling: The server's architecture facilitates modular handling of various requests. Each request type is treated independently, promoting extensibility and the seamless integration of new functionalities. "RES" and "SUCCESS" keywords signify successful execution.

Socket Communication: Sockets establish a reliable channel for communication between client and server. Message exchange, including "RES" and "SUCCESS" indicators, occurs seamlessly in both directions.

Client Interaction: User-driven requests empower clients to navigate functionalities, receiving information from the server. The client initiates requests, and "RES" and "SUCCESS" affirm successful interactions.

In summary, the protocol's implementation features meticulous message organization, a resilient error-handling mechanism, and modular design supporting extensibility. This technical overview provides insight into the architecture governing NFTNet's communication, emphasizing "RES" and "SUCCESS" as key indicators of successful operations.

Conclusion:

The NFTNet project achieves the development of a client-server application, effectively fetching NFT data from the CoinGecko API. The harmonious collaboration of the communication protocol, client, and server components culminates in a cohesive and user-friendly system.