Stock Tracker - Project Documentation

Yosakorn Sirisoot st126512

November, 2024

1 Project Objectives

The main objectives of this project are to design and implement a multi-client stock tracking system with a TCP-based server-client architecture that demonstrates secure, real-time communication. The program allows authenticated users to monitor real-time stock prices, request individual stock data, and receive live updates from a central server.Specifically, the project aims to:

- Develop a server that responds to client requests with structured and standardized messages.
- Establish a TCP connection that allows multiple clients to communicate concurrently with the server.
- Implement user authentication to ensure secure access for authorized users only.
- Enable dynamic stock management by allowing clients to subscribe, unsubscribe, and request stock prices, market news, company information, and dividend data.
- Provide real-time stock price updates to subscribed users for continuous monitoring.
- Use status codes and structured messages to maintain clear and consistent communication between the server and clients.
- Support graceful connection handling and server shutdown notifications to avoid abrupt disconnections.
- Demonstrate a multi-threaded server design that efficiently handles multiple clients simultaneously.

This project integrates networking principles, API usage, and real-time data handling to showcase practical applications of client-server communication in Python.

1.1 Program Purpose

The program allows users to monitor and track real-time stock prices through a TCP client-server system. Authenticated users can request stock prices, subscribe to selected stocks for automatic updates, receive market news, and request company information and dividend data.

Application Characteristics

- Real-time stock monitoring with live updates
- TCP-based reliable communication
- Multi-client support with concurrent connections
- User authentication for secure access
- Stock subscription management (subscribe/unsubscribe)
- Access to live market news
- Company information and dividend data
- Structured responses with status codes
- Graceful connection handling and server shutdown
- Multi-threaded server for efficient client management

2 Source Code

Source Code

The source code for this project, including both the server (`server.py`) and client (`client.py`) implementations, is publicly available on GitHub: `https://github.com/Yosakorn0/StockTrack`

3 Application-Layer Protocol Design

The application layer protocol defines how the client and server communicate, exchange information, and interpret messages in this Stock Tracker System. It establishes a set of rules and conventions to ensure reliable, structured, and meaningful data transfer between multiple clients and the server over a TCP connection.

This protocol encompasses user authentication, stock request handling, subscription management, real-time updates, and status codes. By standardizing commands and responses, the protocol enables clients to interact seamlessly with the server, ensures consistent error handling, and supports multi-client concurrency.

The design of this protocol emphasizes clarity, scalability, and robustness, allowing for future extensions such as additional stock-related commands, enhanced subscription features, or integration with graphical interfaces.

Status Code	**Status Phrase**	**Description**
100	CONTINUE	Request received and processing started.
200	OK	Successful operation.
200	OK (Logout)	Successful logout.
400	BAD$_{R}EQUEST$	Invalid or malformed command
401	UNAUTHORIZED	Invalid username.
404	NOT$_{F}OUND$	Stock ticker not found.
500	SERVER$_{E}RROR$	Server exception occurred.
500	SERVER$_{S}HUTDOWN$	Server is stopping.

4 Implementation and Code Flow

4.1 Server Code Overview

The server manages multiple client connections concurrently and acts as the central hub for retrieving and distributing stock data. It handles user authentication, processes client requests, and delivers real-time stock price updates.

- **User Authentication:** Verifies incoming usernames against an authorized list and denies access to unauthorized users.
- **Process Client Requests:** Handles commands such as requesting live stock prices, subscribing to stocks, unsubscribing, viewing market news, retrieving company information, and dividend data.
- **Maintain Stock Data:** Retrieves live stock prices from the Yahoo Finance API (`yfinance`) and maintains subscription lists for clients.
- **Broadcast Updates:** Sends real-time updates to all clients subscribed to particular stocks, refreshing periodically.
- **Send Status Codes:** Provides structured responses with numeric codes and emojis, such as `200 OK`, `404 NOT_FOUND`, and custom server messages.
- **Manage Connections:** Accepts multiple client connections concurrently using multi-threading and cleans up connections on logout or server shutdown.
- **Server Shutdown Handling:** Gracefully stops the server, notifies connected clients, and closes all active connections safely.

4.2 Client Code Overview

The client connects to the stock server over a TCP connection and serves as the interface for the user to interact with the system. It handles user input, communicates commands to the server, and displays responses or live stock updates.

- **User Authentication:** Sends the username to the server and receives validation; only authorized users can proceed.
- **View Stock Prices:** Requests and displays current prices for individual stocks from the server.
- **Manage Subscriptions:** Allows users to subscribe or unsubscribe to specific stocks, enabling automatic live updates for selected stocks.
- **Display Server Responses:** Shows formatted messages from the server, including stock prices, subscription confirmations, market news, company info, dividend data, and error/status codes.
- **Receive Real-Time Updates:** Continuously listens for broadcasted updates from the server in a background thread without blocking user input.

- **Exit Gracefully:** Sends a logout or exit command and closes the TCP connection cleanly, ensuring no abrupt disconnections.
- **Command Handling:** Supports commands such as `stop`, `back`, and `exit` for intuitive control of live updates and session management.

5 Example Code Explanation

Below is a simple breakdown of key code segments from the client and server files:

5.1 Server Side

```
def handle_client(conn, addr):
    print(f"[CONNECTED] {addr}")
    subscribed = []  # per-client subscription list
    username = "UNKNOWN"
    try:
        # Ask for username
        conn.sendall(b"Enter your username: ")
        username = conn.recv(1024).decode().strip()
        print(f"[DEBUG] Received username: '{username}'")

        # Check authorization
        if username not in AUTHORIZED_USERS:
            send_status(conn, 401)
            conn.close()
            print(f"[DISCONNECTED] Unauthorized user {username}")
            return
        send_status(conn, 200, f"Welcome {username}! Login successful.")
        print(f"[AUTHORIZED] {username} connected from {addr}")

        active = True
        while active:
            menu = (
                "\n===== STOCK MENU =====\n"
                "1. View live stock prices (auto-refresh)\n"
                "2. Subscribe to stocks (continuous updates)\n"
                "3. View latest global market news\n"
                "4. View company info\n"
                "5. View dividend info\n"
                "6. Exit\n"
                "Type 'back' in any submenu to return to this menu.\n"
                "Enter choice: "
            )
            conn.sendall(menu.encode())
            choice = conn.recv(1024).decode().strip().lower()
            if not choice:
                send_status(conn, 400, "No input received")
                continue
```

The server handles client connections by first authenticating the user and sending status codes. Each client has a separate subscription list for stock updates. Unauthorized users are disconnected immediately, while authorized users receive a welcome message and can interact with the server. The server repeatedly sends a stock menu, receives user choices, and responds appropriately, including handling invalid input with a `400 BAD_REQUEST`.

5.2 Client Side

```
def receive_messages(sock):
    while True:
        try:
            data = sock.recv(1024).decode()
            if not data:
                print("\n[SERVER CLOSED CONNECTION]")
                break
            print(data, end='')  # directly print server messages
        except (ConnectionResetError, OSError):
            print("\n[CONNECTION TERMINATED]")
            break
        except Exception as e:
            print(f"\n[ERROR] {e}")
            break
```

This function runs in a background thread, allowing the client to continuously listen for server messages. It prints real-time stock updates, subscription confirmations, and status codes while gracefully handling connection errors.

6 Transport Layer Choice: TCP

This project uses the TCP protocol because it provides **reliable, connection-oriented communication with built-in error handling and acknowledgment.** TCP ensures that all messages—such as stock updates, commands, and responses—are delivered in order and without loss. It maintains persistent sessions between the server and multiple clients, making it ideal for continuous stock updates and command exchanges in a multi-client environment.

Additionally, TCP automatically handles packet loss, retransmission, and data integrity, eliminating the need to implement these mechanisms manually. Its reliability and session management complement the server's multi-threaded architecture and real-time data broadcasting. ion, and data integrity, eliminating the need to implement these mechanisms manually.

7 Conclusion

The Stock Tracker System successfully demonstrates the implementation of a client–server architecture using Python socket programming to deliver real-time financial data. By integrating the Yahoo Finance API, the system provides accurate and up-to-date stock information to authenticated users through a simple command-line interface.

The project highlights the efficiency and reliability of TCP connections for consistent data transmission, supporting multiple concurrent clients and responsive communication. Structured status codes allow users to interpret system responses clearly, enhancing usability and error handling.

Overall, the project meets its objectives of enabling real-time stock monitoring, multi-client support, and secure communication. Future enhancements could include a graphical user interface, chart visualization, or database storage for user preferences to further improve usability and functionality.