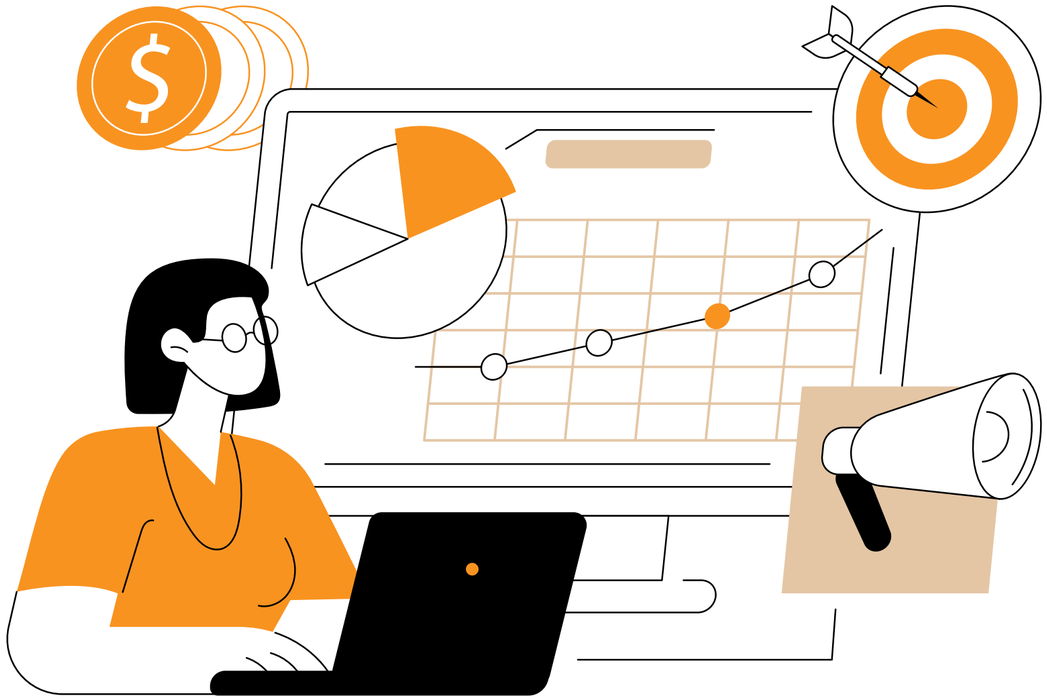
StockTrack

AT70.05 Computer Network



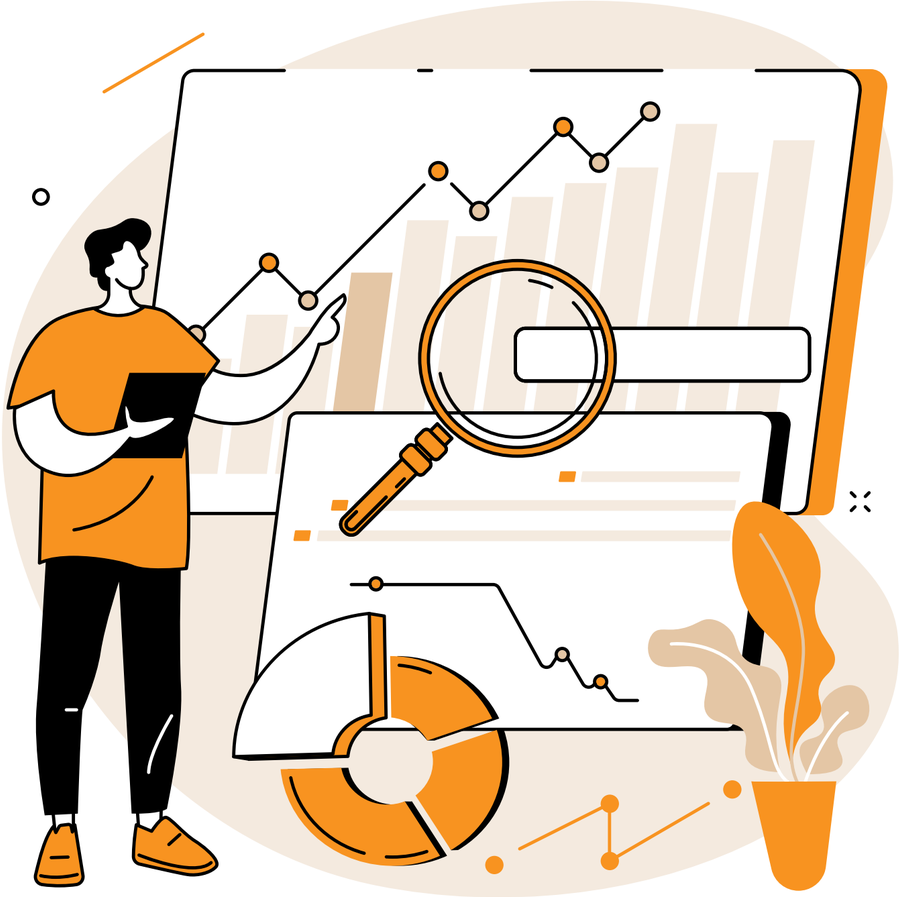
By Yosakorn Sirisoot st126512

Objective

Create a server that responds to client messages with automated responses.

- Establish a TCP-based connection between server and multiple clients.
- Implement user authentication for secure access.
- Allow clients to subscribe, unsubscribe, and request stock prices dynamically.
- Provide real-time stock price updates to subscribed users.
- Use status codes and structured messages for consistent communication.
- Support graceful connection handling and server shutdown notifications.
- Demonstrate multi-threaded server design for concurrent client handling.

Read More



Why TCP?

Reliable Communication

TCP ensures all messages (stock updates, commands, responses) are delivered in order and without loss.

Connection-Oriented:

Maintains a persistent session between the client and server, ideal for continuous stock updates and commands.

Error Handling & Acknowledgment:

Automatically handles packet loss, retransmission, and data integrity, so you don’t need to code that manually.

Server Side

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
```

socket.AF_INET → IPv4
socket.SOCK_STREAM → TCP (stream-oriented, reliable)
s.listen() → server starts listening for incoming TCP connections

```
while server_running:
            try:
                conn, addr = s.accept()
                threading.Thread(target=handle_client, args=(conn, addr)).start()
            except socket.timeout:
                continue
```

This creates a TCP session between server and client.

Client Side

```
1   with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
2           s.connect((HOST, PORT))
3           threading.Thread(target=receive_updates, args=(s,), daemon=True).start()
```

socket.SOCK_STREAM → TCP
s.connect() → establishes a TCP connection to the server

Why not UDP?

No Reliability

UDP does not guarantee message delivery — stock updates or commands could be lost or dropped in transit.

No Order Guarantee

Messages may arrive out of sequence, causing confusion in stock prices or user responses.

No Connection Management:

Messages may arrive out of sequence, causing confusion in stock prices or user responses.

Server Function

User Authentication: Validates username and returns 200 OK or 401 UNAUTHORIZED.

Handle Multiple Clients: Supports simultaneous client connections via multi-threading.

Stock Price Retrieval: Fetches live single or multiple stock prices from Yahoo Finance API.

Subscription Management: Continuously sends updates for subscribed stocks (status 100 CONTINUE).

Command Processing: Handles client commands (view, subscribe, stop, exit) with appropriate status codes.

Session Termination / Logout: Closes client connection and sends 2000 OK (logout).

Error Handling: Catches API/network errors and sends 500 SERVER_ERROR.

Authenticates, fetches data, manage multiple clients, updates subscriptions , handles commands, and returns status codes/errors.

Client Function

For Authenticate User

After connecting to the server and authenticating the user, the client performs two main functions:

View Stock Prices

- The user can request any stock's current price from the list (e.g., GET|AAPL or GET|TSLA).
- The server responds with the latest price and status message.

Manage Subscribed Stocks

- The user can subscribe to certain stocks to receive live updates automatically.
- The user can also view their subscribed stock list using the MY_STOCKS command.

Requests and displays data, manages subscriptions, authenticates.

Client Function

For Authenticate User

After connecting to the server and authenticating the user, the client performs two main functions:

View latest market's news

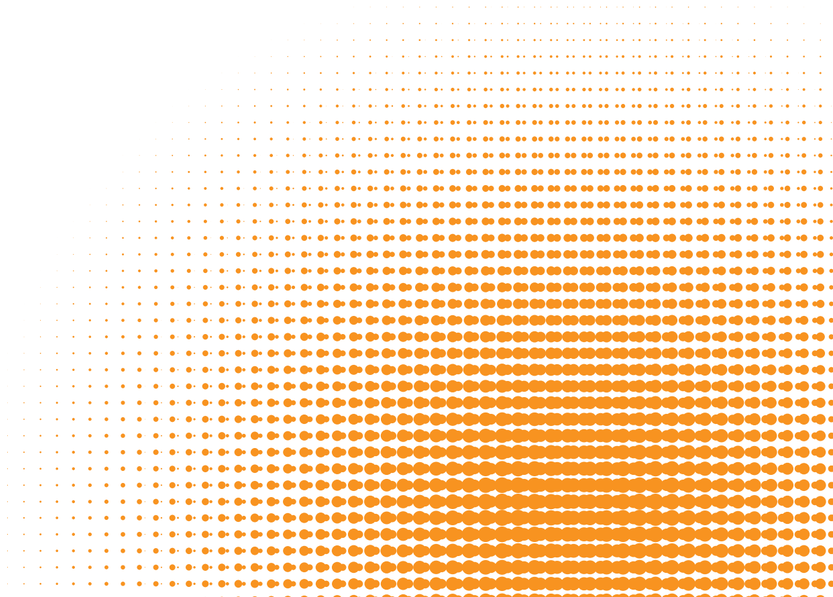
- Aggregates the latest global market headlines from multiple financial RSS feeds and returns the top 10 articles.
- Each news item includes the title, source, and link for easy display to users.

View company information and dividend

- Fetches key company profile details—name, sector, industry, and website—using the Yahoo Finance API.
- Retrieves the five most recent dividend payments for a given stock symbol.
- Converts dividend data into a dictionary and safely handles empty or invalid responses.

SYSTEM ARCHITECTURE

SERVER



Server Side

```python
import socket
import threading
import time
import yfinance as yf
import feedparser

HOST = '127.0.0.1'
PORT = 65432
AUTHORIZED_USERS = ["Yosakorn"]
```

```python
def get_live_price(symbol):
    try:
        stock = yf.Ticker(symbol)
        data = stock.history(period="1d")
        if not data.empty:
            return round(data["Close"].iloc[-1], 2)
        return None
    except Exception:
        return None
```

Server Side

```python
def get_news():
    rss_feeds = [
        "https://www.marketwatch.com/rss/topstories",
        "https://www.cnbc.com/id/100003114/device/rss/rss.html",
        "https://feeds.a.dj.com/rss/RSSMarketsMain.xml"
    ]
    news_items = []
    for url in rss_feeds:
        feed = feedparser.parse(url)
        for entry in feed.entries[:5]:  # top 5 from each feed
            title = entry.get("title", "No title")
            link = entry.get("link", "")
            source = entry.get("source", {}).get("title", url)
            news_items.append((title, source, link))
    return news_items[:10]  # return top 10 combined
```

Server Side

```python
def get_company_info(symbol):
    try:
        stock = yf.Ticker(symbol)
        info = stock.info
        return {
            "Name": info.get("longName", "N/A"),
            "Sector": info.get("sector", "N/A"),
            "Industry": info.get("industry", "N/A"),
            "Website": info.get("website", "N/A")
        }
    except Exception:
        return None
```

Server Side

```python
def get_dividend(symbol):
    try:
        stock = yf.Ticker(symbol)
        div = stock.dividends
        if div.empty:
            return None
        return div.tail(5).to_dict()
    except Exception:
        return None
```

Server Side

```python
def send_status(conn, code, message=""):
    STATUS_CODES = {
        100: "📈 CONTINUE - Command received and being processed",
        200: "✅ OK",
        2000: "🔴 OK (logout) - Successful logout",
        400: "⚠️ BAD_REQUEST - Invalid command",
        401: "❌ UNAUTHORIZED - Invalid username",
        404: "❌ NOT_FOUND - Stock ticker not found",
        500: "💥 SERVER_ERROR - Server-side exception",
        501: "🚧 SERVER_SHUTDOWN - Server is stopping"
    }
    msg = f"{code} {STATUS_CODES.get(code, '')}"
    if message:
        msg += f" | {message}"
    print(f"[SERVER STATUS] {msg}")  # log on server
    try:
        conn.sendall(f"{msg}\n".encode())  # send to client
    except Exception:
        pass
```

Server Side

```python
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}")
```

Server Side

```
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
```

Server Side

```python
if choice == "1":
                conn.sendall(b"Enter stock symbols separated by comma (e.g., AAPL,GOOG): ")
                symbols_input = conn.recv(1024).decode().strip().upper()
                symbols = [s.strip() for s in symbols_input.split(",") if s.strip()]
                if not symbols:
                    send_status(conn, 400, "No valid symbols entered")
                    continue

                conn.sendall(b"Displaying live stock prices every 5 seconds. Type 'stop' or 'back' to return.\n")
                conn.settimeout(5)
                while True:
                    try:
                        for symbol in symbols:
                            price = get_live_price(symbol)
                            if price is not None:
                                conn.sendall(f"📊 {symbol}: ${price}\n".encode())
                            else:
                                send_status(conn, 404, f"{symbol} not found")

                        try:
                            data = conn.recv(1024).decode().strip().lower()
                            if data in ["stop", "exit", "quit"]:
                                send_status(conn, 200, "Stopped live monitoring")
                                conn.settimeout(None)
                                break
                            elif data == "back":
                                send_status(conn, 200, "Returning to main menu")
                                conn.settimeout(None)
                                break
                        except socket.timeout:
                            continue
                    except Exception as e:
                        send_status(conn, 500, str(e))
                        break
```

Server Side

```
elif choice == "2":
                while True:
                    conn.sendall(
                        b"\nEnter a stock symbol to subscribe, 'remove' to remove a stock, or 'done' to start updates: "
                    )
                    symbol = conn.recv(1024).decode().strip().upper()

                    if symbol.lower() == "done":
                        if not subscribed:
                            send_status(conn, 400, "No stocks subscribed yet")
                            continue
                        break
                    elif symbol.lower() == "remove":
                        # Show current subscriptions
                        if not subscribed:
                            send_status(conn, 400, "No stocks to remove")
                            continue
                        conn.sendall(f"Current subscriptions: {', '.join(subscribed)}\n".encode())
                        conn.sendall(b"Enter a stock symbol to remove: ")
                        remove_symbol = conn.recv(1024).decode().strip().upper()
                        if remove_symbol in subscribed:
                            subscribed.remove(remove_symbol)
                            send_status(conn, 200, f"Removed {remove_symbol} from subscriptions")
                        else:
                            send_status(conn, 404, f"{remove_symbol} is not in your subscriptions")
                        continue
                    elif not symbol:
                        send_status(conn, 400, "No symbol entered")
                        continue
                    else:
                        if symbol not in subscribed:
                            subscribed.append(symbol)
                            send_status(conn, 200, f"Subscribed to {symbol}")
                        else:
                            send_status(conn, 200, f"{symbol} already in subscription list")
```

Server Side

```
conn.sendall(f"Updating subscribed stocks every 5 seconds: {', '.join(subscribed)}. Type 'stop', 'back', or 'exit' to return.\n".encode())
                conn.settimeout(5)

                while True:
                    try:
                        for symbol in subscribed:
                            price = get_live_price(symbol)
                            if price is not None:
                                conn.sendall(f"📊 {symbol}: ${price}\n".encode())
                            else:
                                send_status(conn, 404, f"{symbol} not found")

                        try:
                            data = conn.recv(1024).decode().strip().lower()
                            if data in ["stop", "exit", "quit"]:
                                send_status(conn, 200, "Stopped subscription")
                                conn.settimeout(None)
                                break
                            elif data == "back":
                                send_status(conn, 200, "Returning to main menu")
                                conn.settimeout(None)
                                break
                        except socket.timeout:
                            continue
                    except Exception as e:
                        send_status(conn, 500, str(e))
                        break
```

Server Side

```
elif choice == "3":
                conn.sendall("\n📰 Fetching latest global market news...\n".encode())
                news_items = get_news()
                if not news_items:
                    send_status(conn, 404, "No market news found")
                    continue

                for i, (title, publisher, link) in enumerate(news_items, start=1):
                    conn.sendall(f"{i}. {title} — {publisher}\n🔗 {link}\n\n".encode())
                conn.sendall("Type 'back' to return to the main menu.\n".encode())
                try:
                    data = conn.recv(1024).decode().strip().lower()
                    if data == "back":
                        send_status(conn, 200, "Returning to main menu")
                        continue
                except socket.timeout:
                    pass
                send_status(conn, 200, f"Displayed top {len(news_items)} market news articles")
```

Server Side

```
elif choice == "4":
    conn.sendall(b"Enter a stock symbol to view company info: ")
    symbol = conn.recv(1024).decode().strip().upper()
    info = get_company_info(symbol)
    if not info:
        send_status(conn, 404, f"No company info found for {symbol}")
        continue
    for key, value in info.items():
        conn.sendall(f"{key}: {value}\n".encode())
    conn.sendall("Type 'back' to return to the main menu.\n".encode())
    try:
        data = conn.recv(1024).decode().strip().lower()
        if data == "back":
            send_status(conn, 200, "Returning to main menu")
            continue
    except socket.timeout:
        pass
    send_status(conn, 200, f"Displayed company info for {symbol}")
```

Server Side

```
elif choice == "5":
                conn.sendall(b"Enter a stock symbol to view dividend info: ")
                symbol = conn.recv(1024).decode().strip().upper()
                dividends = get_dividend(symbol)
                if not dividends:
                    send_status(conn, 404, f"No dividend data found for {symbol}")
                    continue

                conn.sendall("Date | Dividend\n".encode())
                conn.sendall(("-" * 30 + "\n").encode())
                for date, value in dividends.items():
                    # Ensure date is string
                    if hasattr(date, "date"):
                        date_str = str(date.date())
                    else:
                        date_str = str(date)
                    # Ensure value is float
                    try:
                        value_float = float(value)
                    except Exception:
                        value_float = 0.0
                    conn.sendall(f"{date_str} | ${value_float:.2f}\n".encode())

                conn.sendall("Type 'back' to return to the main menu.\n".encode())

                try:
                    data = conn.recv(1024).decode().strip().lower()
                    if data == "back":
                        send_status(conn, 200, "Returning to main menu")
                        continue
                except socket.timeout:
                    pass
                send_status(conn, 200, f"Displayed dividend info for {symbol}")
```

Server Side

```
            elif choice in ["6", "exit", "quit"]:
                send_status(conn, 2000)
                active = False
                break

            else:
                send_status(conn, 400, "Invalid menu option")

    except Exception as e:
        send_status(conn, 500, str(e))
    finally:
        conn.close()
        print(f"[SESSION CLOSED] {username} from {addr}")
```

Server Side

```
def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.bind((HOST, PORT))
    server.listen()
    print(f"[SERVER RUNNING] Listening on {HOST}:{PORT}")

    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()

```

CLIENT

Client Side

```python
import socket
import threading
import sys

HOST = '127.0.0.1'
PORT = 65432

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

```

Client Side

```python
def start_client():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        try:
            s.connect((HOST, PORT))
            print(f"[CONNECTED] Connected to {HOST}:{PORT}")
            print("Press Enter to continue")
        except ConnectionRefusedError:
            print("Server not available.")
            return

        listener = threading.Thread(target=receive_messages, args=(s,), daemon=True)
        listener.start()

        while True:
            try:
                # No prompt, user just enters
                user_input = input()
                if not user_input:
                    continue

                s.sendall(user_input.encode())

                if user_input.strip().lower() in ["6", "exit", "quit"]:
                    print("[DISCONNECTED] Session terminated.")
                    s.close()
                    sys.exit()

            except (KeyboardInterrupt, EOFError):
                print("\n[CLIENT EXITED]")
                s.close()
                break
            except Exception as e:
                print(f"[ERROR] {e}")
                break
```

Conclusion

- ✅ Built a working client–server stock tracker using Python sockets
- 🧠 Demonstrated key concepts: TCP communication, multi-threading, and real-time updates
- 💬 Implemented status codes for clearer client–server communication
- 🌟 Achieved reliable, real-time stock monitoring for authenticated users

THANK YOU