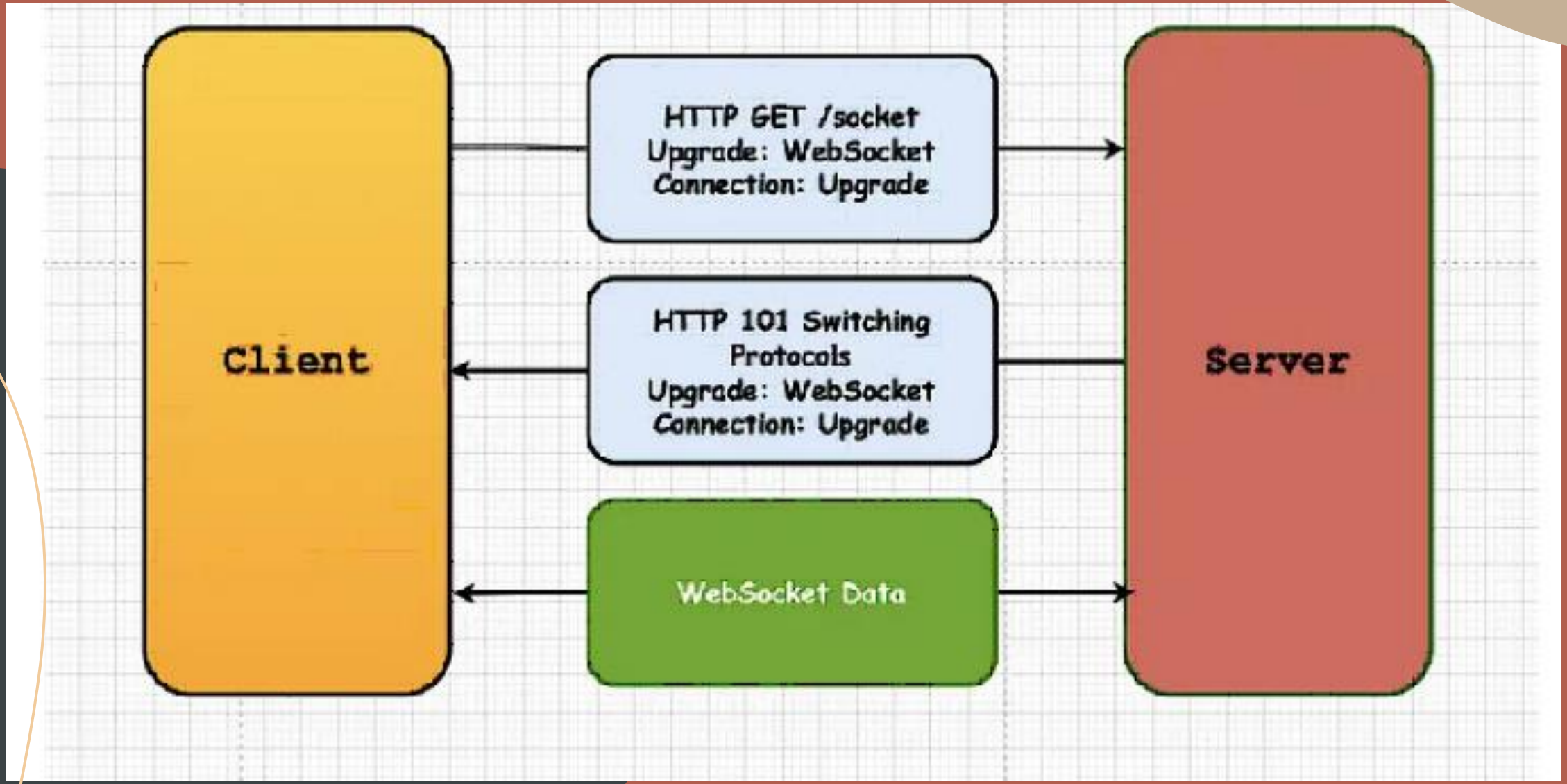
Wipro NAG program –C++ Programming || Linux System programming batch

Capstone project presentation -

Project Title – Online stock market using client and server

Presented by – Deepthi Mashetti


Client
HTTP GET /socket
Upgrade: WebSocket
Connection: Upgrade
HTTP 101 Switching
Protocols
Upgrade: WebSocket
Connection: Upgrade
WebSocket Data
Server

PROJECT OVERVIEW

Create a system where clients can interact with a server to get stock market data, place buy/sell orders, and get updates.

INTRODUCTION

The Real-Time Stock Market System is a client-server application designed to provide a platform for real-time stock trading and market updates. The system allows clients to execute buy and sell orders, retrieve stock prices, and receive updates on stock information.

MOTIVATION

The motivation behind this project is to develop a real-time stock market monitoring system that provides users with up-to-date information on stock prices, trading volumes, and other relevant market data. This system will help investors and traders make informed decisions by providing them with timely and accurate market data.

PROJECT SCOPE

- Develop a client-server architecture using C++ programming language
- Create a server that retrieves real-time stock market data from various sources (e.g. APIs, databases)

- Develop a client that connects to the server and receives real-time market data
- Implement a user-friendly interface for users to view and analyze market data

Various Application Tools That Are Used In This Project:

Networking Tools:

- **Boost.Asio**: Provides a cross-platform C++ library for network and low-level I/O programming, used for asynchronous I/O operations.
- **Socket Programming**: Utilizes BSD sockets for creating network connections between the client and server.

JSON Parsing Tools:

- **JsonCpp**: A library used for parsing and generating JSON data, essential for handling stock market data responses and requests.

•

Concurrency Tools:

- **Threads**: Utilizes C++11 thread library for managing concurrent connections and performing periodic updates of stock prices.
- **Mutex**: Provides synchronization mechanisms to prevent data races when accessing shared resources like stock price data.

System Monitoring Tools:

- **Logging**: Implements logging mechanisms to track server activities, errors, and client interactions.

Build and Compilation Tools:

- **GCC (GNU Compiler Collection)**: Compiles the C++ source code into executable binaries.

Modules That Have Been Worked On In This Project:

Server Module:

- Socket Management: Handles creation, binding, listening, and accepting connections from clients.
- Client Handling: Manages communication with clients, processes requests, and sends responses.

Client Module:

- Request Sending: Sends commands such as "GET_STOCK_PRICES", "BUY <symbol>", and "SELL <symbol>" to the server.
- Response Handling: Receives and displays responses from the server.

Data Handling Module:

- Stock Price Management: Manages stock price data, including updating prices and handling buy/sell operations.
- JSON Integration: Parses and formats JSON data for stock prices and trade requests.

Concurrency and Synchronization Module:

- Thread Management: Implements threading to handle multiple client connections concurrently.
- Mutex Protection: Uses mutexes to ensure thread-safe operations on shared resources.

Error Handling Module:

- Exception Handling: Captures and manages exceptions and errors during network communication and data processing.
- Logging: Logs critical errors and server status updates for debugging and monitoring.

User Interface Module:

Command Line Interface: Provides a command-line interface for clients to interact with the server, issue commands, and view responses.

List of Functions Used In This Project:

Server Functions:

- **Initialization and Cleanup**:
- void startServer():
    - Initializes and starts the server, sets up the listening socket, and accepts incoming client connections.
- void stopServer():
    - Cleans up resources, closes sockets, and terminates server operations.

- **Client Handling Functions**:
- void handleClient(tcp::socket socket):
  - Handles communication with a connected client, processes commands, and sends responses.

Client Functions:

- Connection Management:
  - void connectToServer():
    - Establishes a connection to the server and handles connection errors.

Request and Response Functions:

- void sendRequest(const std::string &request):
  - Sends a request to the server and handles the data transmission.
- std::string receiveResponse():
  - Receives and processes the server's response.

- **Data Handling Functions**:
- **Stock Price Management**:
  - void updateStockPrices():
    - Fetches and updates stock prices from a data source (e.g., stock market API).
  - std::string getStockPrice(const std::string &symbol):
    - Retrieves the current price of a specified stock symbol.
- **Command Processing**:
  - **std::string processCommand(const std::string &command)**:
    - Processes client commands like "GET_STOCK_PRICES", "BUY <symbol>", and "SELL <symbol>", and returns appropriate responses.

- Future Amendments Addressing Common Challenges:

- Handling Increased Load:
  - **Challenge**: As the number of clients increases, the server may need to handle a higher volume of requests and data.
  - **Amendment**: Optimize the server's performance by implementing load balancing, improving concurrency handling, and scaling server resources.
- Extended Functionality:
  - **Challenge**: Users might request additional features such as real-time stock charts, historical data analysis, or advanced trading options.
  - **Amendment**: Extend the system to include additional functionalities like integrating advanced trading APIs, providing historical data analysis, and offering graphical stock visualizations.