# MollyBet Client

## Project Overview

This project implements a client for interacting with the MollyBet API, supporting both HTTP and WebSocket communication. The client handles user authentication via the API, establishes secure WebSocket connections, and processes event messages in real time.

Written using C++20, the codebase follows modern software development principles, emphasizing asynchronous communication, modular design, and robust error handling. Key libraries such as `Boost.Beast` for WebSocket communication and `CPR` for HTTP requests ensure performance and simplicity.

## Key Features

- **HTTP Authentication**: Authenticates with the MollyBet API to retrieve a session token used for WebSocket communication.
- **WebSocket Client**: Establishes a WebSocket connection to listen for event messages and gracefully manages the connection lifecycle.
- **Asynchronous I/O**: Utilizes `Boost.Asio` to handle non-blocking communication for scalability.
- **JSON Processing**: Processes incoming data using `nlohmann::json` for structured message handling.
- **Containerized Deployment**: Includes a Dockerfile for easy containerization and deployment.

## Project Structure

graphql

Copy code

```
.
├── src
│   ├── main.cpp           # Application entry point
│   ├── MollyAPIClient.cpp  # Manages communication with the
MollyBet API
```

```
│   ├── HTTPClient.cpp      # Handles HTTP requests for
authentication
│   ├── WebSocketClient.cpp # Manages WebSocket communication
├── include
│   ├── MollyAPIClient.h    # API client header
│   ├── HTTPClient.h        # HTTP client header
│   ├── WebSocketClient.h   # WebSocket client header
├── Dockerfile              # Docker configuration for building
and running the application
└── README.md               # Project documentation
```

### `main.cpp`

The entry point of the application, responsible for:

- **Initializing Clients**: Creates instances of `MollyAPIClient` and manages the flow between HTTP login and WebSocket event listening.
- **Error Handling**: Catches exceptions and logs errors to ensure the application terminates gracefully in case of failures.
- **Process Flow**: Handles the overall logic of logging in via HTTP, obtaining the session token, and connecting to the WebSocket server to listen for event messages.

### `MollyAPIClient.cpp`

The `MollyAPIClient` class is responsible for interacting with the MollyBet API. It encapsulates the logic needed for API communication, including:

- **Login Management**: Uses the `HTTPClient` to authenticate with the API, sending the username and password to retrieve a session token.
- **Token Handling**: Manages the session token securely, which is passed to the WebSocket client for event stream authentication.
- **Orchestration**: Coordinates the interaction between the `HTTPClient` and `WebSocketClient`, abstracting away the complexity of managing API and WebSocket connections from the main application logic.

This class is designed to act as a high-level API orchestrator, centralizing API interaction logic to simplify usage in the main application.

### `HTTPClient.cpp`

The `HTTPClient` class is a lightweight wrapper around HTTP operations, responsible for sending requests to the MollyBet API:

- **Authentication**: Sends a `POST` request with the user credentials to retrieve the session token.
- **Error Handling**: Validates HTTP responses and checks for errors such as failed logins or malformed JSON responses.
- **JSON Parsing**: Utilizes `nlohmann::json` to extract the session token from the API's response, ensuring the application is authenticated for further WebSocket communication.

### `WebSocketClient.cpp`

The `WebSocketClient` class is responsible for managing WebSocket communication with the MollyBet API:

- **Asynchronous WebSocket Connection**: Uses `Boost.Beast` and `Boost.Asio` to establish a secure WebSocket connection with the API.
- **Error Handling**: Provides comprehensive error checking at every stage of the connection process, including DNS resolution, TCP connection, SSL handshake, and WebSocket handshake.
- **Message Listening**: Listens to incoming messages from the WebSocket stream, processes them as JSON, and handles specific event types (e.g., closing the connection upon receiving a "sync" event).
- **Secure Connection**: The WebSocket connection is encrypted via SSL, and each step in the connection sequence ensures that errors are logged and handled appropriately.

## Design Decisions

1. **Asynchronous Communication**: Given the real-time nature of event messages from the WebSocket, asynchronous I/O was essential. `Boost.Asio` is employed to manage non-blocking I/O for scalability and performance.
2. **Modularity**: The code is structured around clear abstractions (`HTTPClient`, `WebSocketClient`, and `MollyAPIClient`), ensuring that individual components can be easily extended, tested, or replaced without impacting the overall system.

3. **Robust Error Handling**: Throughout the project, care has been taken to handle errors gracefully. Each component logs errors in detail to aid in debugging while ensuring the application doesn't crash unexpectedly.
4. **Security**: All WebSocket communication is encrypted using SSL/TLS, ensuring that data transmitted between the client and the server is secure. The session token is handled carefully, and sensitive information is not exposed in logs.
5. **Docker Deployment**: The project is containerized using Docker, making it easy to deploy in any environment with minimal configuration. This also ensures consistency between development and production environments.

## Running the Project

### Prerequisites

- **C++20** compiler
- **Boost** libraries (for WebSocket and asynchronous I/O)
- **CPR** (for HTTP requests)
- **nlohmann::json** (for JSON processing)
- **Docker** (optional)

### Build Instructions

To compile the project manually:

bash

Copy code

```
mkdir build

cd build

cmake ..

make
```

### Running in Docker

To build and run the project using Docker:

bash

Copy code

```
docker build -t mollybet_client .

docker run mollybet_client
```