Binance Trading System Documentation

Introduction

This project is a Binance Futures API client developed in **C++** using **cURL** for HTTP requests, **OpenSSL** for HMAC SHA256 signature generation, and **JSON** for data parsing. The system enables users to **place market and limit orders, cancel orders, and check the Binance server time**.

This document will provide a **detailed explanation of each part of the code**, an **overview of the working process**, **errors encountered during development**, and **steps taken to resolve them**. It is structured to be **beginner-friendly**, ensuring that even someone with minimal technical knowledge can understand the project.

Project Overview

Features

- Fetches **Binance server time** to ensure accurate request timestamps.
- Places **market and limit orders**.
- Cancels **existing orders**.
- Uses **HMAC SHA256** for secure authentication.
- Implements **cURL** for HTTP requests.
- Handles **API responses and errors gracefully**.

Technologies Used

- **C++** (Core programming language)
- **cURL** (Handles HTTP requests)
- **OpenSSL** (For HMAC SHA256 signature generation)
- **nlohmann/json** (For parsing JSON responses)

Code Walkthrough

1. Include Necessary Libraries

```
#include <iostream>

#include <string>

#include <curl/curl.h>

#include <chrono>

#include <openssl/hmac.h>

#include <iomanip>

#include <sstream>

#include "Include/json.hpp"
```

Explanation:

- iostream: Provides input/output functionality.
- string: Used for handling text data.
- curl/curl.h: Required for making HTTP requests.
- chrono: Used to fetch timestamps.
- openssl/hmac.h: Provides cryptographic functions for signature generation.
- iomanip: Formats output (hexadecimal conversion, padding, etc.).
- sstream: Allows string stream manipulations.
- json.hpp: External JSON library for parsing API responses.

---

2. Function to Handle cURL Response

```
size_t WriteCallback(void* contents, size_t size, size_t nmemb, void* userp) {

  ((std::string*)userp)->append((char*)contents, size * nmemb);

  return size * nmemb;

}
```

Explanation:

- This function **processes API responses** by appending the received data to a string.
- size * nmemb: Calculates the actual response size.
- (char*)contents: Converts received data into a string format.

3. Function to Send HTTP Requests

```
std::string sendRequest(const std::string& url, const std::string& apiKey = "", const std::string& apiSecret = "", const std::string& payload = "", bool isPost = false) {
```

Explanation:

- **URL:** The endpoint to which the request is sent.
- **API Key & Secret:** Used for authentication.
- **Payload:** Holds request parameters.
- **isPost:** Determines whether it is a GET or POST request.

```
  std::string response;
  CURL* curl = curl_easy_init();


  if (!curl) {
    std::cerr << "CURL initialization failed!" << std::endl;
    return "";
  }
```

- Initializes **cURL**.
- If initialization fails, an error message is printed.

```
  curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
  curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
  curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response);
  curl_easy_setopt(curl, CURLOPT_HTTP_VERSION, CURL_HTTP_VERSION_2_0);
  curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
  curl_easy_setopt(curl, CURLOPT_TIMEOUT, 10L);
  curl_easy_setopt(curl, CURLOPT_CAINFO, "C:/Trading System Project/certs/cacert.pem");
```

- **Sets cURL options**, including:
  - URL to send the request to.
  - Callback function to handle responses.
  - HTTP version and timeout settings.

        - SSL certificate path.

---

4. Generating HMAC SHA256 Signature

std::string createSignature(const std::string& data, const std::string& secret) {

- Generates an **HMAC SHA256 signature** using the API secret key.
- Ensures that API requests are **secure and tamper-proof**.

---

5. Placing an Order

void placeOrder(const std::string& apiKey, const std::string& apiSecret, const std::string& symbol, const std::string& side, const std::string& price, const std::string& quantity, const std::string& type) {

- Sends a **trade request** to Binance.
- Uses sendRequest() to interact with Binance API.

---

6. Canceling an Order

void cancelOrder(const std::string& apiKey, const std::string& apiSecret, const std::string& symbol, const std::string& orderId) {

- Cancels an **existing order**.
- Uses Binance **order ID** to identify the order.

---

7. Checking Binance Server Time

void checkBinanceTime() {

- Fetches **Binance server time** to ensure **timestamps are synchronized**.

Errors Encountered and Fixes

1. SSL Handshake Failure

Error: SSL was not OK

- **Fix:** Specified the correct path for cacert.pem.

2. Timestamp Mismatch

Error: timestamp for this request is outside of the recvWindow

- **Fix:** Used Binance server time instead of local system time.

3. Invalid Signature

Error: signature for this request is not valid

- **Fix:** Used proper **HMAC SHA256** signature generation.

4. cURL Request Failure

Error: CURL request failed: Timeout

- **Fix:** Increased timeout and verified internet connection.

Project Setup and Execution

1. Install Dependencies

- Install **cURL**.
- Install **OpenSSL**.
- Install **nlohmann/json.hpp**.

2. Compile the Program

Use the following command to compile:

```
g++ -o binance_trading main.cpp -lcurl -lssl -lcrypto
```

3. Run the Program

```
./binance_trading
```

Conclusion

This Binance Trading System successfully allows users to place, cancel orders, and retrieve server time with secure authentication. With proper API integration, it can be expanded to include more trading functionalities such as **order book analysis and automated trading bots**.

This documentation ensures that even beginners can **understand, modify, and enhance** the project with ease. 🚀