Documentation

Lakshay Bansal
lakshay21059@iiitd.ac.in

Source code documentation

Trader:

Methods:

```
std::pair<int, std::string> place_order(const std::string&, double, int);
std::pair<int, std::string> cancel_order(const std::string&);
std::pair<int, std::string> modify_order(const std::string&, double, int);
std::pair<int, std::string> get_orderbook(const std::string&, int);
std::pair<int, std::string> view_position(const std::string&);
std::pair<int, std::string> get_openorders(const std::string&);
std::pair<int, std::string> get_marketdata(const std::string&, int);

int handleCancelOrder(const std::function<std::pair<int, std::string>(std::string)>& action);
int handlePlaceOrder(const std::function<std::pair<int, std::string>(std::string, double, int)>& action);
int handleModifyOrder(const std::function<std::pair<int, std::string> (std::string, double, int)>& action);
int handleGetOrderBook(const std::function<std::pair<int, std::string> (std::string, int)>& action);
int handleViewPosition(const std::function<std::pair<int, std::string> (std::string)>& action);
int handleOpenOrders(const std::function<std::pair<int, std::string> (std::string)>& action);
int handleMarketData(const std::function<std::pair<int, std::string>(std::string, int)>& action);
```

Member:

```
Api *m_api;
```

test_latency / test_throughput:

Methods:

```
void place_order(int &orders, Api& api);
void place_order_async(int& orders, Api& api);
void clear_orders(Api& api);
```

Member:

```
Api api = Api();
```

Socket:

Methods:

```
virtual ~Socket();
virtual void switch_to_ws() = 0;
[[nodiscard]] virtual std::pair<int, std::string> ws_request(const std::string& msg) = 0;
virtual void ws_request_async(const std::string& msg, std::function<void(int, const std::string&)> callback) = 0;
virtual void ws_response_async(int status, const std::string& resp) = 0;
```

Member:

```
const std::string host = "test.deribit.com";
const std::string port = "443";
std::vector<std::pair<int, std::string>> m_response_buffer;
```

BSocket:

Methods:

```
BSocket(); // Constructor
~BSocket(); // Destructor
void switch_to_ws() override;
[[nodiscard]] std::pair<int, std::string> ws_request(const std::string& msg) override;
void ws_request_async(const std::string& msg, std::function<void(int, const std::string&)> callback) override;
void ws_response_async(int status, const std::string &resp) override;
void connect_async();
void on_resolve(const boost::system::error_code& ec,
boost::asio::ip::basic_resolver<boost::asio::ip::tcp>::results_type results);
void on_connect(const boost::system::error_code& ec);
void on_handshake(const boost::system::error_code& ec);
```

Member:

```
net::io_context ioc_async;
ssl::context ctx_async;
tcp::resolver resolver_async;
websocket::stream<beast::ssl_stream<tcp::socket>>* m_ws;
websocket::stream<beast::ssl_stream<tcp::socket>>* m_ws_async;
std::thread m_io_thread;
bool async_flag = false;
bool q_empty = true;
boost::asio::io_context::work work_guard;
```

Socketpp:

Methods:

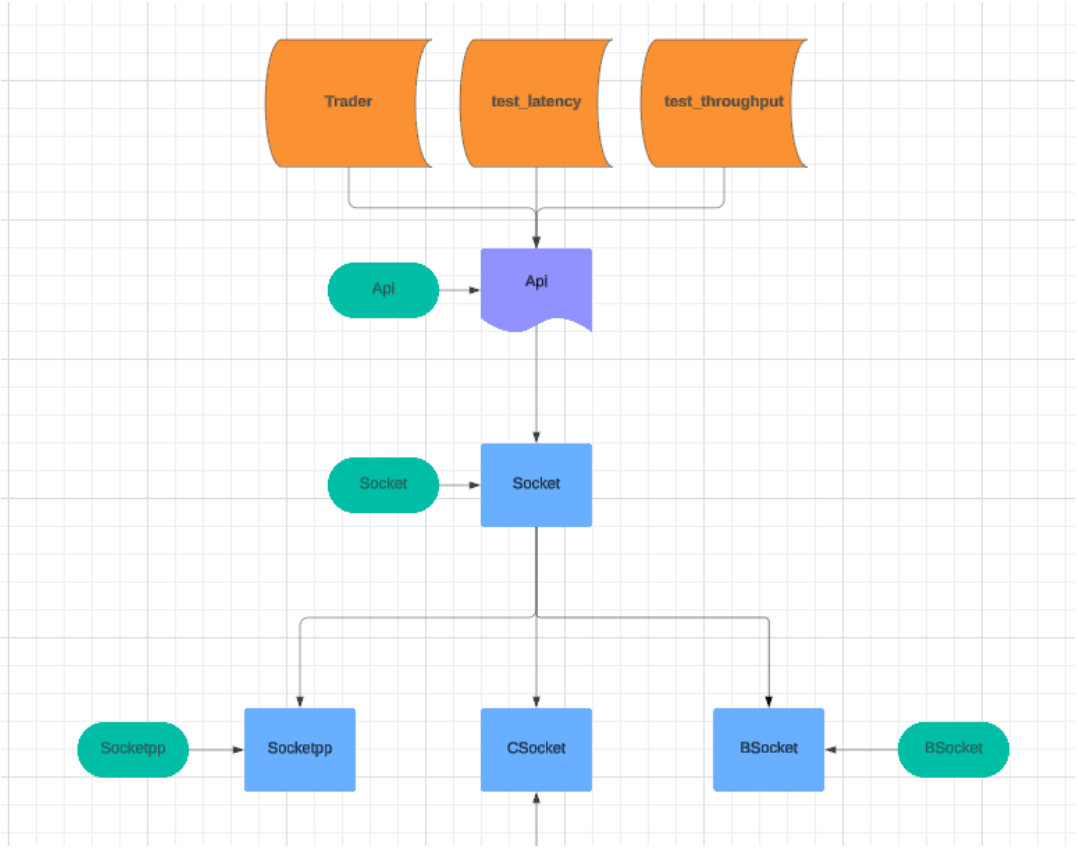
```
Socketpp(); // Constructor
~Socketpp(); // Destructor
void switch_to_ws() override;
[[nodiscard]] std::pair<int, std::string> ws_request(const std::string& msg) override;
void ws_request_async(const std::string& msg, std::function<void(int, const std::string&)> callback) override;
void ws_response_async(int status, const std::string& resp) override;
```

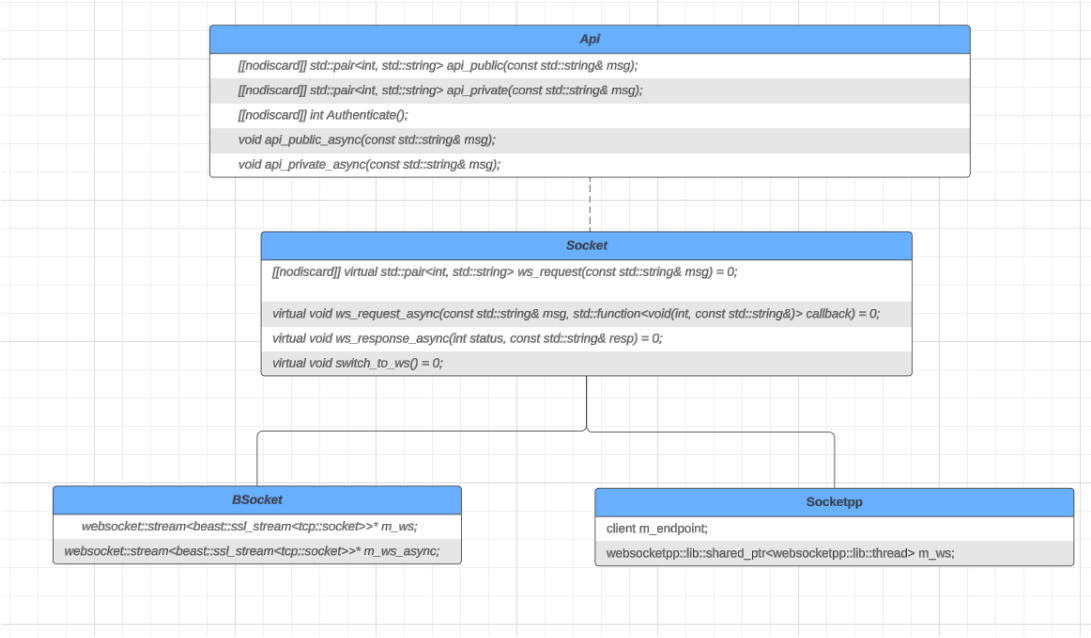
Member:

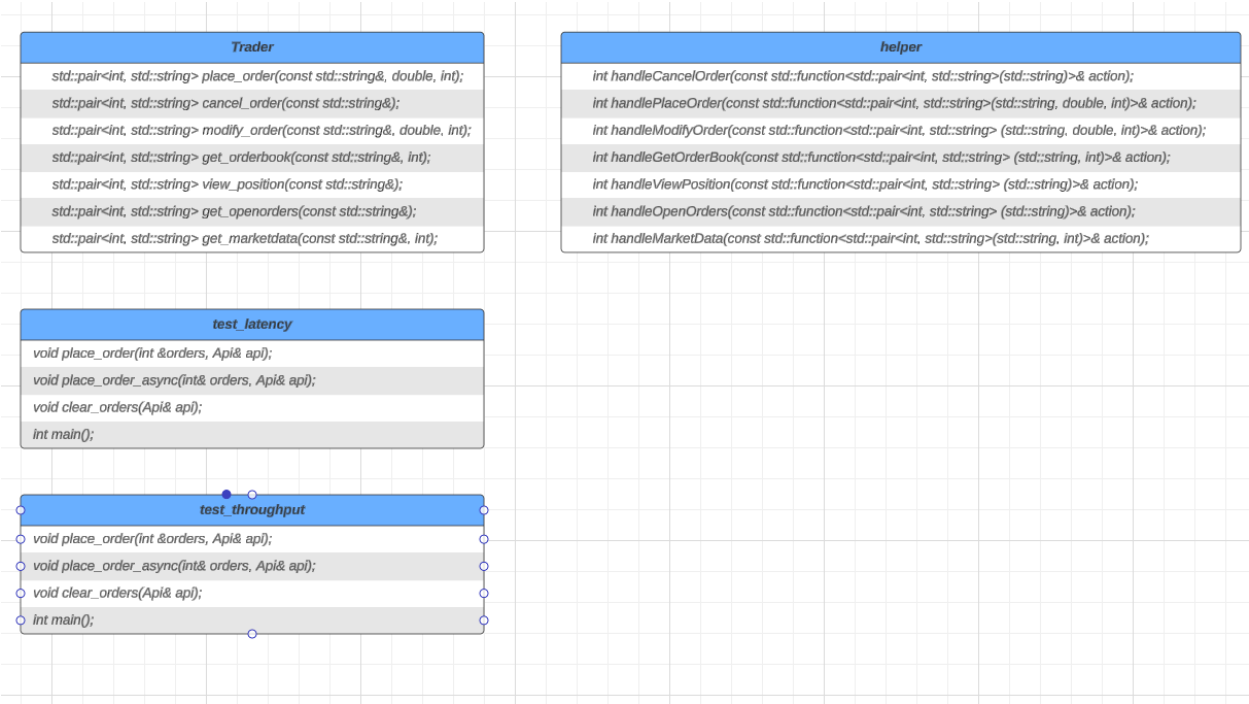
```
client m_endpoint;
connection_metadata::ptr con_metadata;
websocketpp::lib::shared_ptr<websocketpp::lib::thread> m_ws;
```

*also a class id defined in the header file

```
class connection_metadata {
public:
	typedef websocketpp::lib::shared_ptr<connection_metadata> ptr;
	connection_metadata(int id, websocketpp::connection_hdl hdl, std::string uri) : m_id(id)
	, m_hdl(hdl)
	, m_status("Connecting")
	, m_uri(uri)
	, m_server("N/A") {}
	connection_metadata() {}

	void on_open(client * c, websocketpp::connection_hdl hdl) {
	m_status = "Open";
		std::cout << "Connection opened!\n";
	client::connection_ptr con = c->get_con_from_hdl(hdl);
	m_server = con->get_response_header("Server");
	}

	void on_fail(client * c, websocketpp::connection_hdl hdl) {
	m_status = "Failed";
		client::connection_ptr con = c->get_con_from_hdl(hdl);
	m_server = con->get_response_header("Server");
	m_error_reason = con->get_ec().message();
		std::cout << "Connection failed: " << m_error_reason << "\n"; // Log failure reason
	}

	void on_message(websocketpp::connection_hdl /*hdl*/, client::message_ptr msg){
		{
		std::lock_guard<std::mutex> lock(m_mutex);
		msg_queue.push_back(msg->get_payload());
		}
		m_cv.notify_one(); // Notify one waiting thread
	}
public:
	int m_id;
	websocketpp::connection_hdl m_hdl;
	std::string m_status;
	std::string m_uri;
	std::string m_server;
	std::string m_error_reason;
	std::vector<std::string> msg_queue;
	std::mutex m_mutex;
	std::condition_variable m_cv;
};
```

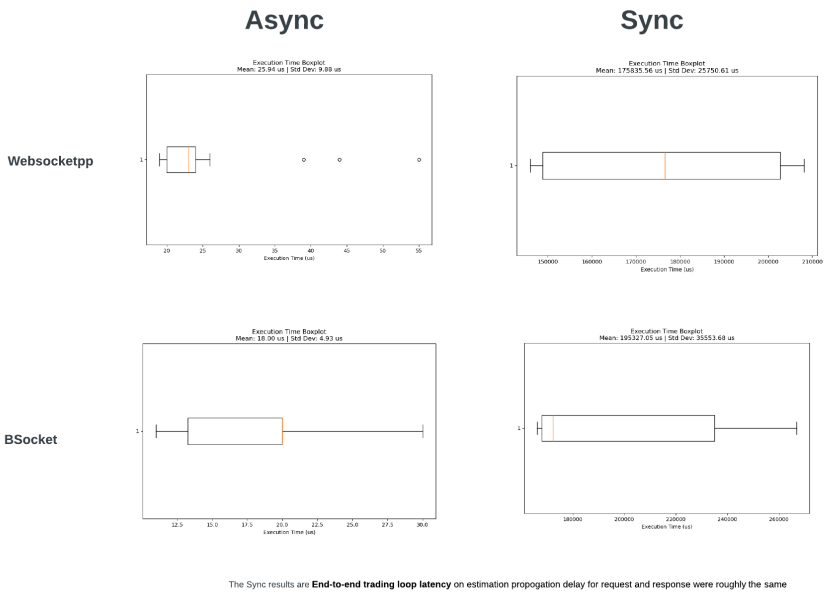

Trader
test_latency
test_throughput
Api
Api
Socket
Socket
Socketpp
Socketpp
CSocket
BSocket
BSocket


Api
[[nodiscard]] std::pair<int, std::string> api_public(const std::string& msg);
[[nodiscard]] std::pair<int, std::string> api_private(const std::string& msg);
[[nodiscard]] int Authenticate();
void api_public_async(const std::string& msg);
void api_private_async(const std::string& msg);
Socket
[[nodiscard]] virtual std::pair<int, std::string> ws_request(const std::string& msg) = 0;
virtual void ws_request_async(const std::string& msg, std::function<void(int, const std::string&)> callback) = 0;
virtual void ws_response_async(int status, const std::string& resp) = 0;
virtual void switch_to_ws() = 0;
BSocket
websocket::stream<beast::ssl_stream<tcp::socket>>* m_ws;
websocket::stream<beast::ssl_stream<tcp::socket>>* m_ws_async;
Socketpp
client m_endpoint;
websocketpp::lib::shared_ptr<websocketpp::lib::thread> m_ws;



Bonus Section:

Findings: Async calls performed better than their synchronous counterparts. In case of Websocket++ and Boost Beast Socket I found out my Websocket++ implementation outperformed BSocket or (Boost implementation).



The Sync results are **End-to-end trading loop latency** on estimation propogation delay for request and response were roughly the same

Optimizations

1. Use of char literals for requests and response (minimal use of nlohmann/json library):
- I figured out that using the library for parsing may slow down the application so Just for showcasing results to the user in case of a trader the library is used to pretty print the response.
2. Passing by reference wherever possible to avoid unnecessary copies made during transfer of objects.
3. Proper Benchmarking to get a better idea regarding what are the bottlenecks.

Future Work:

1. Continue Developing my own Socket library tailor made for our needs.
2. Rectify the bugs or issues in the code.

Repo link: https://github.com/Talkative-Banana/Low-Latency-Trading-System