The code is compiled using the commands:

```
# from project root
cmake -B build
cmake --build build -j$(nproc)
```

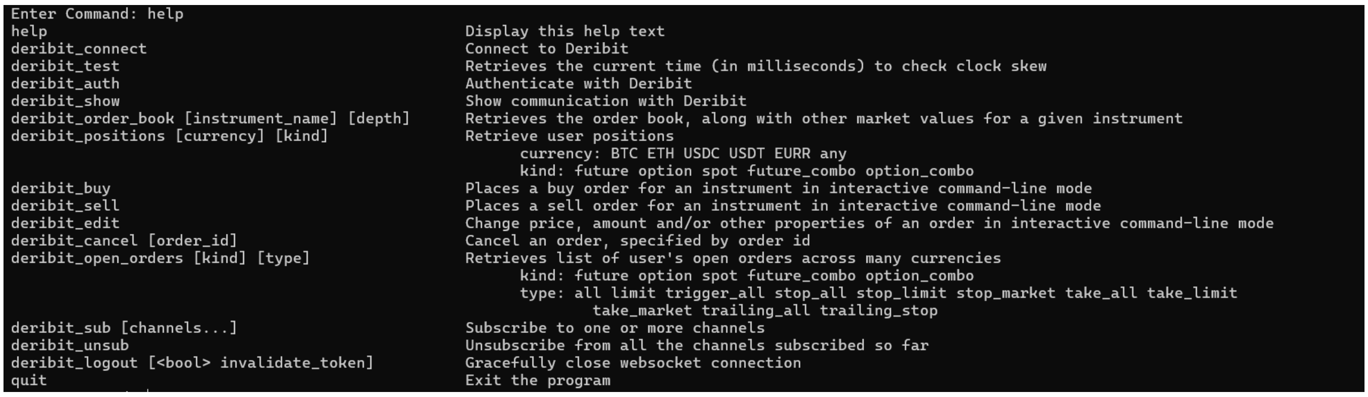
It has been tested on WSL2 Ubuntu. The executable will be created in the project root as `client_trader`.

Code Structure

The structure of the code is as follows:

```
src/
│   client_main.cpp
│
├───api
│       deribit.h
│       trade_handler.h
│
├───client
│       client_trader.cpp
│       client_trader.h
│
├───lib
│       benchmark.h
│       utilities.h
│
└───websocket
        websocket.cpp
        websocket.h
    api_key.json
    client_main.cpp
CMakeLists.txt
```

- `client_main.cpp` - This file contains the main function. It controls the command loop, with the commands as follows:



```
Enter Command: help
help                                            Display this help text
deribit_connect                                 Connect to Deribit
deribit_test                                    Retrieves the current time (in milliseconds) to check clock skew
deribit_auth                                    Authenticate with Deribit
deribit_show                                    Show communication with Deribit
deribit_order_book [instrument_name] [depth]    Retrieves the order book, along with other market values for a given instrument
deribit_positions [currency] [kind]             Retrieve user positions
                                                      currency: BTC ETH USDC USDT EURR any
                                                      kind: future option spot future_combo option_combo
deribit_buy                                     Places a buy order for an instrument in interactive command-line mode
deribit_sell                                    Places a sell order for an instrument in interactive command-line mode
deribit_edit                                    Change price, amount and/or other properties of an order in interactive command-line mode
deribit_cancel [order_id]                       Cancel an order, specified by order id
deribit_open_orders [kind] [type]               Retrieves list of user's open orders across many currencies
                                                      kind: future option spot future_combo option_combo
                                                      type: all limit trigger_all stop_all stop_limit stop_market take_all take_limit
                                                              take_market trailing_all trailing_stop
deribit_sub [channels...]                       Subscribe to one or more channels
deribit_unsub                                   Unsubscribe from all the channels subscribed so far
deribit_logout [<bool> invalidate_token]        Gracefully close websocket connection
quit                                            Exit the program
```

- `lib/utilties.h` provides helpful functions for logging, as well as customizing the log stream by using flags. `lib/benchmark.h` provides a class for benchmarking as discussed in the latency report.

```
// Logging Format
// [<time>] <log type>: <msg>
enum class log_flags {
    none          = 0,
    ws            = 1,
    client_trader = 2,
    // ...
};

constexpr static log_flags ENABLED_LOG_FLAGS = (log_flags::ws |
log_flags::client_trader ... )

// Example usage
APP_LOG(log_flags::ws, "Error sending message: " << ec.message());
```

- `websocket/websocket.{h,cpp}` provide declarations and definitions for the application's websocket endpoint.
    - It involves a `connection_metadata` class to store the communication between the client and the server.
    - The `websocket_endpoint` class holds all methods relevant to websockets communication, such as `connect()`, `send()`, `on_message()` etc. It is done using the websocketspp library
- `client/client_trader.{h,cpp}` - This class acts as the interface for all trading methods which the user can invoke. It contains a pointer to the instance of the trade handler which the user passes. This is commonly called the **dependency-inversion pattern**

```
std::unique_ptr<deribit> deribit_uptr = std::make_unique<deribit>();
trade_handler* deribit_handler = deribit_uptr.get();

client_trader trader {deribit_handler, key}; // pass
```

- `api/trade_handler.h` - It is a common interface for `client_trader` class to invoke trade API methods. It is an abstract class from which a class must derive to define the API methods and endpoints.
- `api/deribit.h` - It contains all the logic relevant to trading on deribit platform. The class `deribit` derives from `trade_handler`.
- `api_key.json` - Contains the API id and secret for Deribit.

Libraries used

- websocketspp
- nlohmann json