EURONEXT PROJECT

DEVELOPMENT OF TRADING INTERFACE

Group 3 - Trading Manager (Trader)

Antoine GUILLAUME

Mihaja RANDRIAMAROMANAN

Valery WAH

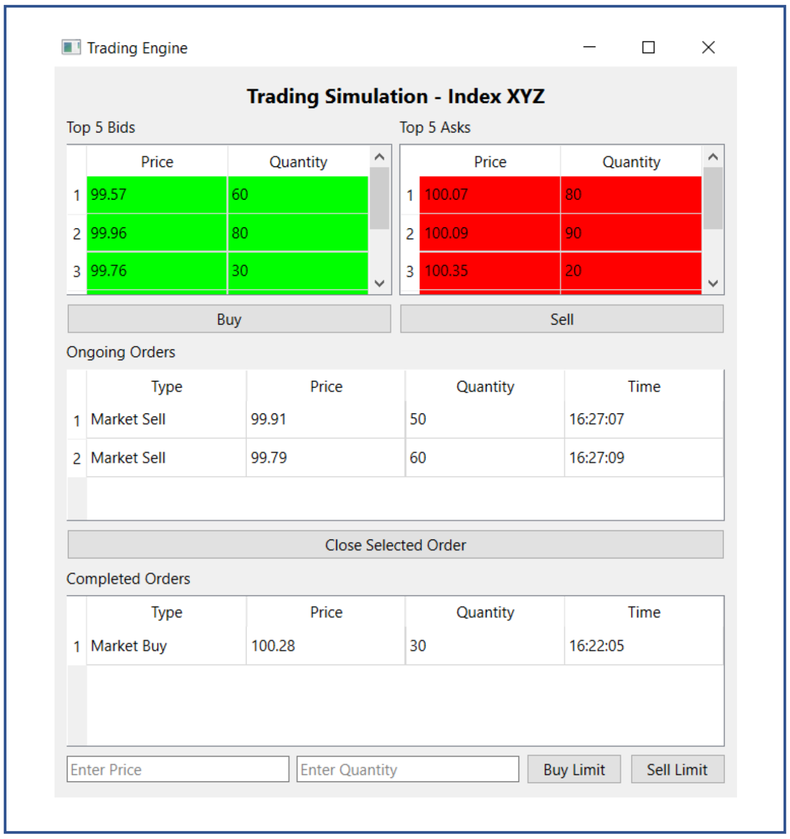
Yacine DRISS

Prince Berthault FOWANG SEGNOU

03/06/2025

Introduction

In the ever-evolving landscape of financial markets, traders are increasingly reliant on sophisticated digital tools to make informed decisions. Speed, precision, and clarity of information are paramount. As part of our project, we have developed an initial prototype of a trading interface tailored to the needs of a professional trader. This report details our design choices, technical implementation, and potential areas for future development.


Trading Engine

Trading Simulation - Index XYZ

Top 5 Bids

	Price	Quantity
1	99.57	60
2	99.96	80
3	99.76	30

Top 5 Asks

	Price	Quantity
1	100.07	80
2	100.09	90
3	100.35	20

Buy Sell

Ongoing Orders

	Type	Price	Quantity	Time
1	Market Sell	99.91	50	16:27:07
2	Market Sell	99.79	60	16:27:09

Close Selected Order

Completed Orders

	Type	Price	Quantity	Time
1	Market Buy	100.28	30	16:22:05

I. Understanding the trader's needs

Before diving into the technical aspects, it was crucial to define what makes a trading interface truly effective. A trader needs immediate access to real-time market data, a clear order book display, seamless order execution, and insightful historical data. Our goal was to craft an interface that not only meets these needs but also enhances the trading experience through efficiency and usability.

To achieve this, we structured our interface around three core components:

- **Live market data display**: real-time updates of bid and ask prices.
- **Order book management**: intuitive visualization of open and completed orders.
- **Database integration**: secure and efficient storage of order history.

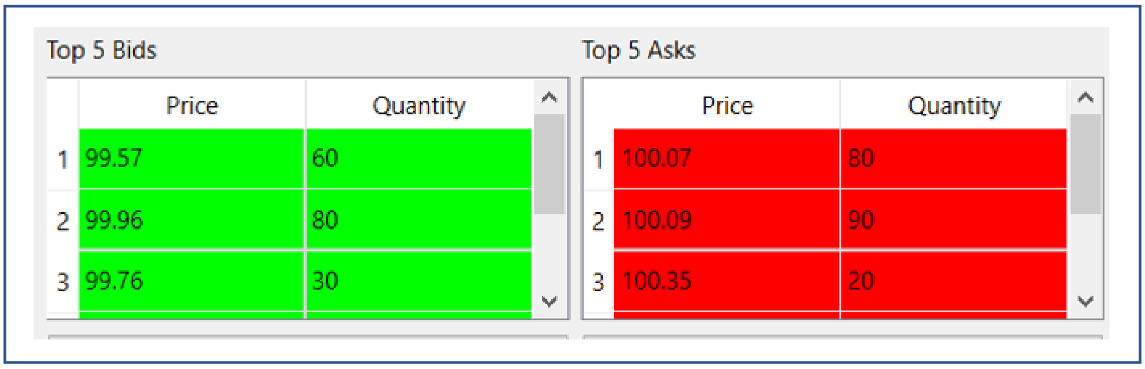
Our group specifically focused on the **display aspect** of the interface, ensuring that all necessary trading information is presented clearly and efficiently. However, the project is still in progress, and the next step will involve integrating our display system with the work of another group responsible for connecting the database (hosted on MongoDB) and linking all system components together.

Overview of our trading interface

Our **Trading Simulation - Indice XYZ** interface has been designed to facilitate a **realistic and structured trading experience**, focusing on **order book visualization, trade execution, and historical tracking**.

A/ Order book section (Top 5 Bids & Top 5 Asks)

The order book is split into **two primary tables**, displaying market liquidity:



Top 5 Bids

	Price	Quantity
1	99.57	60
2	99.96	80
3	99.76	30

Top 5 Asks

	Price	Quantity
1	100.07	80
2	100.09	90
3	100.35	20

Top 5 Bids (Buy orders) - Left table (Green)

- Lists the highest bid prices along with the available quantity.
- The **highest bid price appears at the top**, ensuring traders can instantly identify **buying demand**.
- Color-coded **green**, enhancing readability.
- Example entries from the screenshot:
  - **99.57** → 60 units
  - **99.96** → 80 units
  - **99.76** → 30 units

Top 5 Asks (Sell orders) - Right table (Red)

- Lists the lowest ask prices along with the available quantity.
- The **lowest ask price appears at the top**, ensuring traders can immediately see **selling pressure**.
- Color-coded **red**, making it easy to differentiate from bids.
- Example entries from the screenshot:
  - **100.07** → 80 units
  - **100.09** → 90 units
  - **100.35** → 20 units

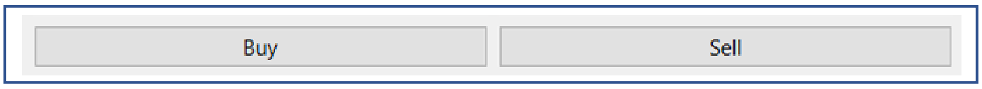
This bid-ask structure allows traders to quickly assess market **depth, liquidity, and price spread** before making trading decisions.

B/ Trade execution buttons

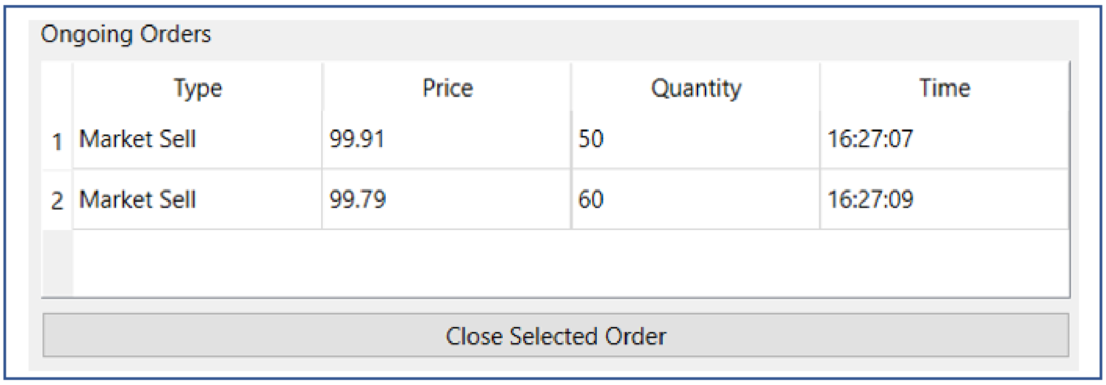
Below the bid and ask tables, we have **two primary action buttons**:

- **"Buy" button**: Executes a market **buy** order by selecting the best (lowest) ask price available.
- **"Sell" button**: Executes a market **sell** order by selecting the best (highest) bid price available.

These buttons mimic **real-world market orders**, ensuring that trades are executed instantly using the **current best available price**.



C/ Ongoing orders section


Ongoing Orders

	Type	Price	Quantity	Time
1	Market Sell	99.91	50	16:27:07
2	Market Sell	99.79	60	16:27:09

This section provides a **real-time summary of active orders** that have been executed but are still open. It consists of:

- **Type** (Buy/Sell)
- **Price** (Execution Price)
- **Quantity** (Number of units traded)
- **Time** (Execution timestamp)

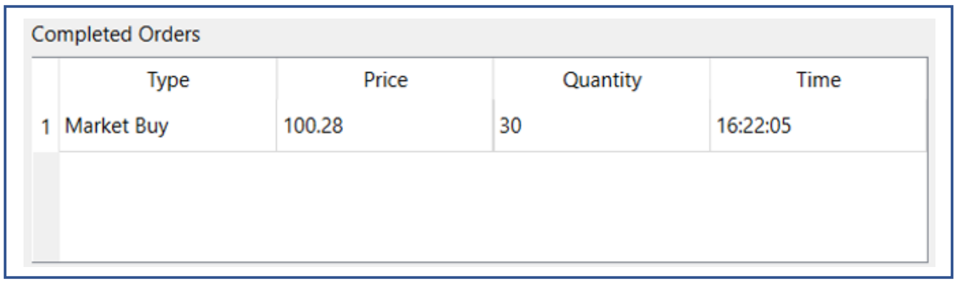
Entries from the screenshot

Type	Price	Quantity	Time
Market Sell	99.91	50	16:27:07
Market Sell	99.79	60	16:27:09

These entries indicate that two **market sell orders** were executed recently, at different prices and quantities.

At the bottom of this section, we have a **"Close Selected Order"** button, which allows users to **finalize and move orders** from the ongoing orders table to the completed orders table.

D/ Completed orders section


Completed Orders

	Type	Price	Quantity	Time
1	Market Buy	100.28	30	16:22:05

This section maintains a **historical record of executed trades**, ensuring traders can track their activity over time. It consists of:

- **Type** (Buy/Sell)
- **Price**
- **Quantity**
- **Time**

Example entry from the screenshot

Type	Price	Quantity	Time
Market Buy	100.28	30	16:22:05

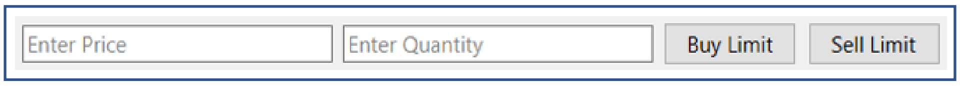
This entry shows that a **market buy order** was executed at **100.28**, for **30 units**, at **16:22:05**.

By keeping a **distinction between ongoing and completed orders**, the interface ensures clear **trade lifecycle management**.

E. Limit order input & execution

At the bottom of the interface, traders can **place limit orders** using:

- **Two input fields**
  - **Enter Price** → Allows users to specify the price for a limit order.
  - **Enter Quantity** → Allows users to specify the order size.
- **Two execution buttons**
  - **"Buy Limit"** → Places a limit buy order at the specified price.
  - **"Sell Limit"** → Places a limit sell order at the specified price.



Unlike market orders, **limit orders remain in the order book** until the market price reaches the specified level.

F. Functional and visual considerations

- **Color coding**
  - **Green for bids**, **red for asks** → Makes it easy to interpret data at a glance.
- **Dynamic UI updates**
  - A QTimer refreshes the order book every **2 seconds**, simulating real-time market fluctuations.
- **Structured layout**
  - Organized with QVBoxLayout and QHBoxLayout for **clarity and usability**.
- **Realistic trading mechanics**
  - Market orders execute instantly.
  - Limit orders allow for **strategic** price execution.
  - The order lifecycle is tracked through the **Ongoing** and **Completed Orders** sections.

→ To launch the **trading interface**, follow these steps to ensure a smooth execution of the program:

1. **Navigate to the build directory:**
   Before running the application, ensure that you are inside the correct build directory where the project has been compiled. Open a terminal or command prompt and execute the following command:

```
cd build
```

This step ensures that you are in the correct directory where all compiled binaries and CMake-generated files reside.

2. **Compile the project:**
   Once inside the build directory, you need to compile the project using CMake. Run the following command:

```
cmake --build . --config Debug
```

This command instructs CMake to build the project using the specified configuration (Debug mode in this case). If the build completes successfully, the executable file **TradingInterface** will be generated in the build folder.

3. **Run the trading interface:**
   Once the build process is completed, you can launch the trading application by executing:

```
./TradingInterface
```

This command starts the graphical user interface, allowing you to interact with the trading engine. Ensure that all required dependencies, such as Qt libraries, are properly installed and accessible in your system path.

III. Code structure and functionalities

Translating these requirements into a functional system required a robust and scalable architecture. Our implementation is built using Qt6, a framework well-suited for high-performance desktop applications.

Frontend design: the interface is designed using Qt Widgets to ensure an intuitive and responsive user experience.

Database management: we utilized SQLite to store and manage trading data efficiently.

Real-time data handling: a QTimer refreshes market data at regular intervals to provide an accurate representation of market conditions.

Each of these components was carefully developed to balance performance with usability, ensuring that the system remains responsive even under high-frequency trading conditions. However, for full integration, our system must be connected to the MongoDB database managed by the other team, allowing for real-time data updates from external sources.

There are the following key components implemented:

DatabaseManager class: handles database connectivity, order insertion, retrieval, and status updates.

Order processing logic: enables smooth execution of buy and sell orders.

Graphical Interface: displays market trends, order book updates, and transaction history in real time.

Real-time order book management: The interface dynamically updates bid and ask prices every 2 seconds using a QTimer to refresh data, ensuring traders receive the most current market conditions.

Order execution mechanics: Market orders (instant buy/sell) remove the best available bid or ask from the order book, mimicking real-world trading behavior.

Limit order placement: The UI provides input fields for traders to specify price and quantity for limit buy/sell orders, allowing for more strategic trading.

Order history management: A QTableWidget stores ongoing and completed trades, including timestamps for accurate record-keeping.

User-friendly layout: The interface is structured with QVBoxLayout and QHBoxLayout, ensuring an intuitive and clear presentation of trading data.

Detailed breakdown of the interface sections

1. **Order book display (Bids & Asks)**
    - A QTableWidget displays the top 5 bid (buy) and ask (sell) orders.
    - Bid orders are color-coded in green, while ask orders are in red for quick identification.
    - Data is refreshed dynamically with a function that generates randomized market movements.

2. **Trade execution mechanism**

   - A “Buy” button executes a market buy order, selecting the best (lowest price) ask available.
   - A “Sell” button executes a market sell order, selecting the best (highest price) bid available.
   - The executed orders are moved into the “Ongoing Orders” table.

3. **Ongoing and completed orders management**

   - The ongoing orders table keeps track of trades that are yet to be finalized.
   - A “Close Order” button allows users to move an ongoing order into the completed orders table, simulating trade finalization.

4. **Limit orders handling**

   - Users can place custom buy and sell limit orders by entering a price and quantity in input fields.
   - A limit order remains in the order book until it is executed at the specified price.

5. **Real-time data synchronization**

   - A QTimer triggers data updates every 2 seconds, ensuring bid and ask prices fluctuate naturally over time.
   - The system uses an event-driven approach to refresh the UI without significant performance overhead.

➔ To ensure smooth interaction between these components, we have implemented clean separation between data handling and UI rendering, following best practices in software engineering. Once our interface is fully connected to the MongoDB database and external trading systems, it will provide seamless updates and accurate trading data representation.

IV. Challenges and future improvements

No development process is without its challenges. One of the primary difficulties encountered was ensuring real-time synchronization between the database and the UI. A delay in updating the order book could mislead the trader, leading to suboptimal decisions. To mitigate this, we optimized our database queries and implemented an efficient event-driven update system.

Future integration will introduce new challenges, particularly in **latency management**. Handling real-time data streams from MongoDB while ensuring a smooth user experience will require efficient data processing techniques. **Network delays, database query response times, and UI rendering speed** will need to be carefully optimized to maintain a responsive interface.

Another challenge was designing an interface that remains user-friendly despite handling complex financial data. To tackle this, we incorporated dynamic resizing, intuitive color coding, and clear tabular layouts to enhance readability.

This prototype lays the groundwork for a more comprehensive trading system. Several enhancements can be planned for future iterations, including:

- **Integration with external market feeds**: Connecting to APIs that provide real-time financial data from major exchanges.
- **Advanced charting tools**: Implementing interactive candlestick charts and technical indicators.
- **Automated trading strategies**: Adding algorithmic trading capabilities based on predefined strategies.
- **Optimization of data handling**: Implementing caching mechanisms and efficient query management to reduce database-related latency.
- **Scalability considerations**: Ensuring that the interface can handle a high volume of trades and multiple data sources simultaneously.

With continuous refinement and innovation, this project has the potential to become a valuable asset. The next crucial step will be working closely with the database and integration team to ensure seamless connectivity between the frontend display and backend data processing. This collaboration will determine the final success of the trading interface, ensuring traders can rely on **accurate, real-time, and well-structured market data** to make their trading decisions with confidence.

Conclusion

Developing an interface that meets the expectations of professional traders is both a technical and conceptual challenge. This initial prototype demonstrates the feasibility of an efficient, real-time trading platform and sets the stage for future improvements. As we move forward, our focus will remain on refining usability, enhancing data processing, and expanding the system's capabilities to create a truly state-of-the-art trading tool.

This project has been an invaluable experience for our team. Beyond the technical challenges, it has been a journey into the **precision and efficiency required in financial technology**. We have seen how **code is not just a sequence of instructions ; it is the bridge between thought and execution, the means through which ideas take shape and become functional reality**.

A trader relies on clarity, speed, and precision, and our interface strives to reflect these values. But on a deeper level, this project has been a testament to how **coding is more than a skill - it is a language, a mindset, a way of constructing solutions that shape the world**.

Just as a trader reads the markets, interprets signals, and makes strategic decisions, a developer reads problems, analyzes structures, and builds solutions. The act of writing code is not just about logic - it is about creating, innovating, and bringing abstract concepts to life.

We extend our gratitude to everyone who contributed to this project. From brainstorming sessions to debugging marathons, every challenge has been an opportunity to grow. This is just the beginning - what we have built today lays the foundation for even greater advancements tomorrow. The market waits for no one, and neither does innovation.