Specification Document - Trading Manager - Listing Module

Efrei Paris x Euronext
Final Project C++

Group no1
Group Members: DAVOINE Amaury, BERTHAULT Alexandre, BOUCHER Mayeul, CLOTTEAU Mathis

March, 2025

Version History

Date	Version	Description	Author
18 November, 2024	1.0	Initial draft	Amaury, Alexandre, Mathis, Mayeul
04 December, 2024	1.1	Added the architecture and the project structure	Amaury, Alexandre, Mathis, Mayeul
31 January, 2025	1.2	Added the Project Development State to track progress	Amaury, Alexandre, Mathis, Mayeul
March 2025	1.3	Added the final project description and implementation details	Amaury, Alexandre, Mathis, Mayeul

Contents

1 Introduction

1.1 Context

The Trading Manager is a core module in the trading system, responsible for managing and supervising various trading functionalities. The **listing module** plays a key role by managing financial instruments available for trading. It ensures that the instruments meet the required standards and provides accurate, up-to-date data for trading activities.

1.2 Objective

The Listing Module will provide functionalities to add, update, delete, and validate financial instruments, ensuring that all trading activities comply with the rules defined by the system and the market.

2 Project Requirements

2.1 Domain-Specific Terminology

- **ISIN:** The ISIN code (International Securities Identification Number) is a unique key that is linked to each financial instrument, e.g. bonds, stocks, options etc... It is formed by two letters for each country (FR for France, UK for United Kingdom), and ten numbers for each instrument.
- **MIC:** The MIC (Market Identification Code) is a four character code that is unique for each trading venue.

2.2 Functional Requirements

2.2.1 Login

The system must allow users to log in using their credentials, ensuring secure access to the platform. It should validate the username and password and provide appropriate error messages if the credentials are incorrect.

The system must be able to retrieve and assign user roles based on the logged-in user's credentials. The roles will define the level of access and permissions granted to each user within the system.

2.2.2 Add a Financial Instrument

- **Description:** This function allows users to add a new financial instrument to the system. We will have to determine which user can be authorized to perform this action.
- **Input:** ISIN code, MIC, trading currency, authorized price fluctuation, lot size, number of decimals, User ID.
- **Validation Rules:**
  - ISIN must be **unique**.
  - MIC must be **EXACTLY** four alphanumeric characters to respect ISO 10383.

  - Trading currency must be valid (e.g., USD, EUR, etc...).
  - Lot size must be a **positive** integer
  - Authorized price fluctuation and decimals must match market standards and be realistic
  - User ID must match the authorized User IDs

- **Output:** The instrument is added to the database and becomes visible in the Trading Manager interface. A validation message can be seen on the GUI.

2.2.3 Modify a Financial Instrument

- **Description:** This function allows users to update the details of an existing instrument.
- **Editable Fields:** Lot size, authorized price fluctuation, trading currency, etc...
- **Non-Editable Fields:** ISIN code, MIC.
- **Validation Rules:**
  - Trading currency must be valid (e.g., USD, EUR, etc...).
  - Lot size must be a positive integer
  - Authorized price fluctuation and decimals must match market standards and be realistic
  - User ID must match the authorized User IDs
- **Output:** The instrument's updated details are saved and reflected in the system. A validation must be seen on the GUI.

2.2.4 Delete an Instrument

- **Description:** This function allows users to delete an instrument in the database
- **Output:** The instrument is removed from the active trading list. A validation must be seen on the GUI.

2.2.5 View the List of Instruments

- **Description:** Displays a list of all active instruments with their attributes (e.g., ISIN, MIC, price fluctuation).
- **Filters:** Users can filter by currency, status (active/inactive), search for a specific value (e.g. ISIN, MIC), etc...
- **Output:** The list of instruments is displayed on the GUI.

2.3 Non-Functional Requirements

2.3.1 User Interface

The interface for adding and viewing instruments must be user-friendly.

2.3.2 Reliability

The code must ensure that invalid data (e.g., duplicate ISINs, negative lot sizes) cannot be added to the database.

2.3.3 Security

Only **authorized** users should be able to add, modify, or delete instruments.

2.4 System Integration

2.4.1 Scheduler Module

Once an instrument is listed, it can be scheduled for trading sessions. Therefore, Surveillance will need to access our data, such as ISIN and MIC.

2.4.2 Pricer Module

Our Module provides the instrument's price fluctuation limits and decimals, which are used to calculate valid prices, thus the Surveillance module needs our data.

2.4.3 Trader Module

The trader will need to access to everything before placing an order. Thus, we need to interact with them.

2.4.4 GUI

Provides a graphical interface for users to interact with all the functionalities of the listing Module (e.g., add, view, and modify instruments). We need to align our GUI with the other modules.

3 System Architecture

3.1 High-Level Design

The system follows a client-server architecture with the following key components:

- RESTful API server built with Crow C++ library
- SQLite database for persistent storage
- Qt-based graphical user interface
- Authentication and authorization system for secure access

3.2 API Architecture

3.2.1 Authentication and Authorization Endpoints

- **POST /login**: Authenticates users and provides access tokens
- **GET /user/role**: Verifies user roles and permissions

3.2.2 Instrument Management Endpoints

- **GET /instruments**: Retrieves list of all available instruments
- **POST /instrument**: Adds new instruments to the system
- **PUT /instrument**: Updates existing instrument details
- **DELETE /instruments/isin**: Removes instruments from the database

3.3 Server Architecture

- **SQLite Database**: Stores user, role, and instrument data
- **API Server**: Built using Crow C++ library for handling HTTP requests
- **Docker Container**: Ensures consistent deployment across environments

3.4 GUI Architecture

- **Authentication/Login Page**: Allows users to enter credentials for authentication with appropriate validation.
- **List of Active Instruments Page**: Displays all active instruments with their attributes, search functionality, and management buttons.
- **Add New Instrument Page**: Provides input fields for adding new instruments with validation.

4 Development Timeline

4.1 November 2024 - Initial Planning

- Completed initial requirements analysis
- Designed the fundamental system architecture
- Established project structure and responsibilities

4.2 December 2024 - Architecture Design

- Finalized the API design and endpoints
- Designed the database schema
- Established the GUI framework approach

4.3 January 2025 - Mid-Project Progress

- Implemented the core API functionality
- Created initial wxWidgets GUI
- Created user authentication system with different role types
- Made decision to transition to Qt GUI

4.4 February-March 2025 - Final Implementation

- Replaced wxWidgets with Qt-based interface
- Merged interface with Surveillance Development Group
- Expanded user authentication for multiple roles
- Dockerized the API server for cross-platform compatibility
- Conducted integration testing with other modules

5 Technical Implementation

5.1 API Implementation

5.1.1 Crow Library Integration

The API was built using the Crow C++ library, which provides:

- Efficient routing for HTTP requests
- Middleware support for authentication and validation
- Built-in JSON handling for structured data exchange
- Support for all HTTP methods for RESTful API design

5.1.2 Database Design

SQLite was chosen for its lightweight, serverless nature, making it ideal for this application where a relational database system is required without the overhead of a full-fledged database server.

5.2 Docker Implementation

To make the API server accessible across different operating systems and avoid dependency issues, we containerized the server using Docker. This ensures that anyone can run the server without worrying about installing libraries or managing environment configurations.

5.2.1 Docker Setup Instructions

1. Download and install Docker Desktop from the official website.
2. Clone the repository for the API from GitHub:

   ```
   https://github.com/Mayeul78/EuronextAdminGUI
   ```

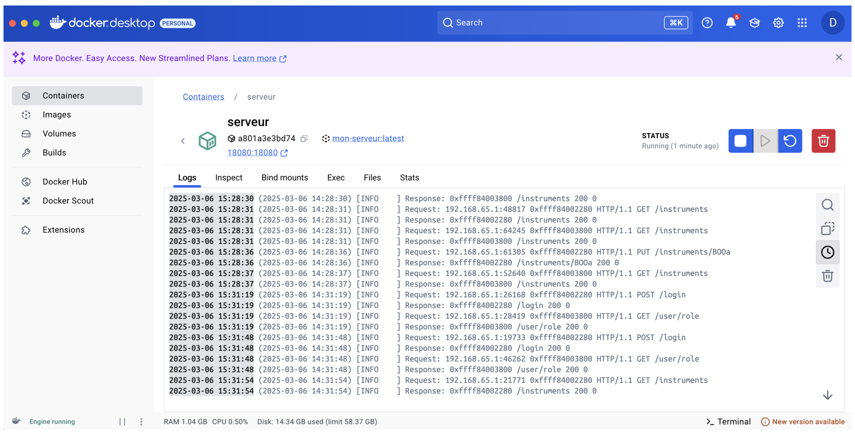
3. Navigate to the directory of the project in your terminal or command prompt.
4. Build the Docker image using the following command:

```
docker build -t mon-serveur .
```

5. After building the image, run the Docker container with the following command:

   ```
   docker run -d --name serveur -p 18080:18080 mon-serveur
   ```

   This will start the API server inside a Docker container, accessible at `http://localhost:18080` and logs can be viewed on Docker Desktop.



```
2025-03-06 15:28:30 (2025-03-06 14:28:30) [INFO    ] Response: 0xffff84003800 /instruments 200 0
2025-03-06 15:28:31 (2025-03-06 14:28:31) [INFO    ] Request: 192.168.65.1:48817 0xffff84002280 HTTP/1.1 GET /instruments
2025-03-06 15:28:31 (2025-03-06 14:28:31) [INFO    ] Response: 0xffff84002280 /instruments 200 0
2025-03-06 15:28:31 (2025-03-06 14:28:31) [INFO    ] Request: 192.168.65.1:64245 0xffff84003800 HTTP/1.1 GET /instruments
2025-03-06 15:28:31 (2025-03-06 14:28:31) [INFO    ] Response: 0xffff84003800 /instruments 200 0
2025-03-06 15:28:36 (2025-03-06 14:28:36) [INFO    ] Request: 192.168.65.1:61305 0xffff84002280 HTTP/1.1 PUT /instruments/BOOa
2025-03-06 15:28:36 (2025-03-06 14:28:36) [INFO    ] Response: 0xffff84002280 /instruments/BOOa 200 0
2025-03-06 15:28:37 (2025-03-06 14:28:37) [INFO    ] Request: 192.168.65.1:52640 0xffff84003800 HTTP/1.1 GET /instruments
2025-03-06 15:28:37 (2025-03-06 14:28:37) [INFO    ] Response: 0xffff84003800 /instruments 200 0
2025-03-06 15:31:19 (2025-03-06 14:31:19) [INFO    ] Request: 192.168.65.1:26168 0xffff84002280 HTTP/1.1 POST /login
2025-03-06 15:31:19 (2025-03-06 14:31:19) [INFO    ] Response: 0xffff84002280 /login 200 0
2025-03-06 15:31:19 (2025-03-06 14:31:19) [INFO    ] Request: 192.168.65.1:28419 0xffff84003800 HTTP/1.1 GET /user/role
2025-03-06 15:31:19 (2025-03-06 14:31:19) [INFO    ] Response: 0xffff84003800 /user/role 200 0
2025-03-06 15:31:48 (2025-03-06 14:31:48) [INFO    ] Request: 192.168.65.1:19733 0xffff84002280 HTTP/1.1 POST /login
2025-03-06 15:31:48 (2025-03-06 14:31:48) [INFO    ] Response: 0xffff84002280 /login 200 0
2025-03-06 15:31:48 (2025-03-06 14:31:48) [INFO    ] Request: 192.168.65.1:46262 0xffff84003800 HTTP/1.1 GET /user/role
2025-03-06 15:31:48 (2025-03-06 14:31:48) [INFO    ] Response: 0xffff84003800 /user/role 200 0
2025-03-06 15:31:54 (2025-03-06 14:31:54) [INFO    ] Request: 192.168.65.1:21771 0xffff84002280 HTTP/1.1 GET /instruments
2025-03-06 15:31:54 (2025-03-06 14:31:54) [INFO    ] Response: 0xffff84002280 /instruments 200 0
```

Figure 1: API server on Docker

5.3 GUI Implementation

5.3.1 Qt Framework Integration

After initially developing with wxWidgets, we transitioned to Qt for:

- Better cross-platform compatibility
- Integration with the surveillance team's existing interface
- More modern UI components and styling options
- Improved development workflow

5.3.2 User Interface Features

The Qt interface implements:

- Role-based access control
- Instrument management screens
- Search and filtering capabilities
- Validation feedback for user actions

6 Usage Instructions

6.1 Starting the Server

1. Follow the Docker setup instructions in section 5.2.1
2. Verify server is running at `http://localhost:18080/ping`

6.2 Running the GUI

1. Download the interface from GitHub: `https://github.com/ThomasRondeau/Euronext-Surveil` `tree/Mayeul`
2. Ensure you are using the `Mayeul` branch
3. Open the project with Qt Creator by selecting the file `QT_LineChart.pro`
4. Compile and run the project
5. If you modify the server's port or address, ensure that the changes are also applied in the `main` file

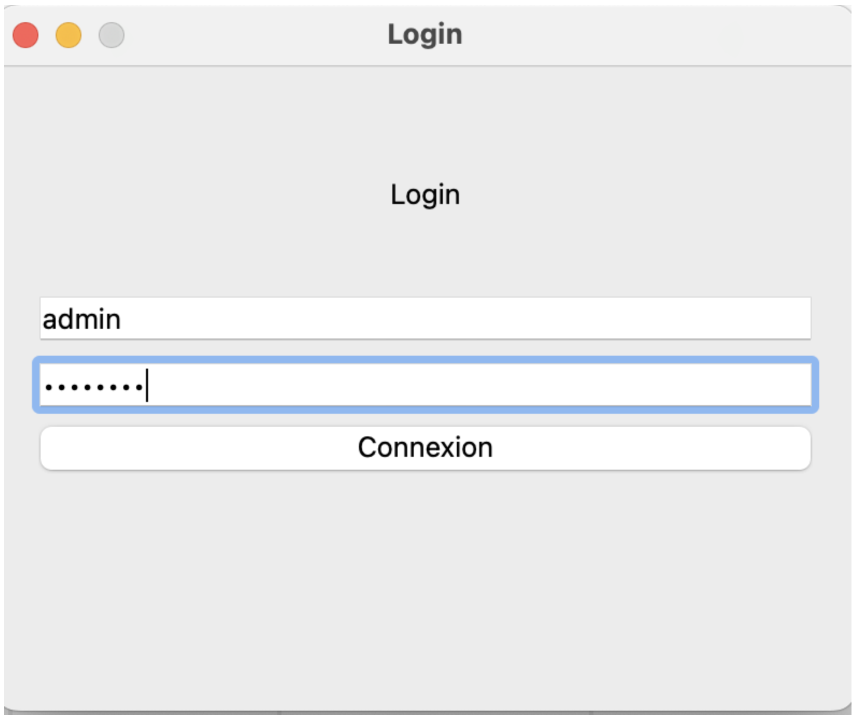
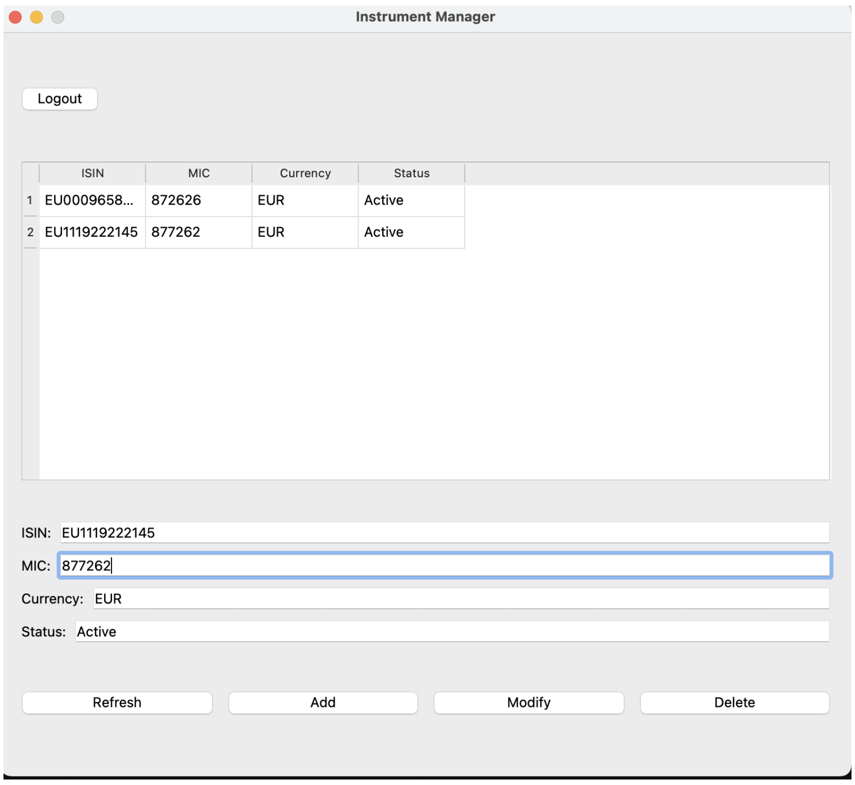


Figure 2: Login Page on the GUI


Instrument Manager

Logout

	ISIN	MIC	Currency	Status
1	EU0009658...	872626	EUR	Active
2	EU1119222145	877262	EUR	Active

ISIN: EU1119222145

MIC: 877262

Currency: EUR

Status: Active

Figure 3: Listing and Modifictation of Instuments

7 Conclusion

7.1 Summary of Achievements

The Listing Module provides a complete solution for managing financial instruments in the trading system, including:

- Secure user authentication and role-based access
- Comprehensive instrument management capabilities
- Integration with other system modules
- Cross-platform compatibility through Docker containerization
- Modern user interface with Qt that could be build for multiple os

7.2 Future Improvements

Potential enhancements for future iterations:

- Advanced filtering and search options
- Connection to the matching Engine was not possible as they were not ready until after February vacations

8 Resources

8.1 Project Repositories

- **API Github:** `https://github.com/Mayeul78/EuronextAdminAPI`
- **WX GUI Github:** `https://github.com/Mayeul78/EuronextAdminGUI`
- **QT GUI Github with surveillance part:** `https://github.com/ThomasRondeau/Euronext-Surveillance/tree/Mayeul`

8.2 Contact Information

For questions regarding the Project: mayeul.boucher@efrei.net