Universitat de Lleida

TREBALL FINAL DE GRAU

ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Jaume Guasch Llobera

Titulació: Doble Grau en Enginyeria Informàtica i ADE

Títol de Treball Final de Grau: Design and Implementation of a Financial Operations Management System

Director/a: Sílvia Maria Miquel Fernández i Jordi Virgili Gomà

Presentació

Mes: Setembre

Any: 2024

Index

1. Introduction

This project focuses on the development of a web application designed to register and manage financial derivatives from scratch. The idea for this project originates from my experience as a Computer Science Engineer at a company that relied heavily on these financial tools. During my time there, I observed two previous attempts to build similar solutions, both of which fell short of meeting the expectations of the financial teams who would have become the users if put into production. The challenges that the development teams found motivated me to take on the project myself, bringing in new ideas with the goal of creating a solution that would not only work for my former company but also be versatile enough to serve a wider range of businesses and corporations.

Financial derivatives play a critical role in today's global economy, particularly in managing the risks associated with fluctuating prices of commodities, stocks, currencies, and more. However, the complexity of these instruments often necessitates advanced tools for effective management. While trading companies sometimes provide the necessary tools for handling these operations, a centralized solution would be ideal for businesses that work with multiple traders, allowing them to consolidate and control their operations more effectively on a single platform. The goal of this project is to develop a user-friendly web application that simplifies these processes and makes them more accessible while also discussing the best approach to gathering requirements, designing, implementing, and ultimately testing the solution.

A key feature of this application is its ability to automate many of the calculations and tasks involved in managing financial derivatives. By providing real-time information, analytics, and data that can be used for accounting purposes, the software will help reduce the burden of manual tasks that can lead to errors or inaccuracies.

The proposed solution consists of a robust backend to manage server-side logic and database interactions, and a dynamic frontend to ensure a seamless and responsive user experience. Integration with external systems, such as those used by trading companies or clients' business software (such as an ERP or an accounting software, for instance), will be a key priority, alongside designing the interface to be accessible and user-friendly across various devices. On a broader level, the project will include an explanation of the application's practical use, illustrating how it streamlines operations, reduces the risk of human error, and provides valuable insights that can be used for better decision-making.

Through this work, I aim to contribute to the field of financial technology (FinTech) by offering a practical tool that helps businesses manage financial derivatives more effectively. The project itself covers a vast range of software development processes and will help grow my own abilities to develop software on a more personal level.

2. Background and motivation

In all sorts of industries, companies routinely use financial tools as integral components of their operational strategies. These financial instruments play a crucial role in various aspects of business, including investment strategies aimed at maximizing profits, hedging against price fluctuations, and managing market risks. For businesses that deal with commodities and other assets, derivatives help offset the risks associated with market volatility and fluctuations in prices.

Due to the complexity of these instruments and the high volume of transactions involved, it is essential to have a robust and efficient system for tracking and managing these trades, especially from a management perspective. Effective management ensures accurate financial reporting, compliance with regulatory requirements, and valuable insights into trading performance. Without such a system, companies may face challenges in maintaining clear records, analyzing financial outcomes, and making informed decisions overall.

During my work at a company that made extensive use of a variety of financial tools, I noticed that managing the vast number of transactions across various markets and traders presented a significant challenge. Accurate registration of all trades is crucial not only for accounting and regulatory compliance but also for performance analysis and informed decision-making. Therefore, detailed records are necessary to understand the sources of profits and losses and to make strategic adjustments based on these insights.

However, my experience highlighted that the existing solutions in use at the time, such as Microsoft Excel, even though simple and cost-less, were inadequate for meeting these purposes. The high volume of transactions, the variety of operations, and the complexity of calculations quickly overwhelms Excel's capabilities. Additionally, the simultaneous data entry by multiple users and the importance of sending this data to another software made it challenging to maintain data integrity and produce accurate reports. The limitations of these type tools also impeded the ability to generate reliable, timely, and precise reports for management. In most cases, management prefers a fast, accessible, and accurate solution to get a clear overview rather than dealing with data manually, which often translates into reduced efficiency, less control over the operations and increased risk of human error.

Driven by these challenges, I decided to start this project to develop a web application designed to tackle these issues. At the same time, I aimed to explore the best approaches, technologies, and methodologies to effectively achieve this goal.

3. Time Allocation Overview and Planning

A thorough overview of the time allotted to each important project phase is provided by the Gantt chart that is being displayed in the following page. It starts with requirements collecting, where time is spent interacting with the stakeholders on-site to fully comprehend their needs and record necessary criteria. This first stage is essential for laying the groundwork for the project's future course and guaranteeing that all stakeholder expectations are well-defined. In June 2023, the project was left on hold, but most of the requirements gathered from the users have had an implication on the development of this project.

During the first stage, while development had been taking place for three months and the project was left on hold, the decision was taken my myself to dig further in this topic in the two thesis that I have carried out during 2024: the Business Management Bachelor's Thesis and the Computer Science Engineering Thesis. It was necessary to work on the existing requirements to make a more ambitious project that could work for multiple organizations, markets and different financial products.

After the on-site meetings, the project has moved into a research phase with the goal of improving the requirements that were gathered and expanding the team's understanding of market dynamics and financial management. To guarantee that the solution is both current with industry insights and customized to the needs of the customers, this step is essential, and the requirement from the different stakeholders is consolidated with the newer requirements of the project that have been discovered during 2024 and discussed in the Business Administration's Thesis.

At last, there is a development phase, where the solution is created to satisfy the given requirements. This timeline is represented by a Gantt chart, which offers an organized method of overseeing the project's development and guarantees that every stage is given the proper attention and resources.

3.1. Project Timeline and Allocation

	TASK TITLE	START DATE	DUE DATE	PHASE ONE				PHASE TWO				PHASES THREE AND FOUR		
				OCT - NOV '22	DEC '22 - JAN '23	FEB - MAR '23	APR - MAY '23	MAR '24	APR '24	MAY '24	JUN '24	JUL '24	AUG '24	SEPT' '24
1	Requirement gathering on-site													
1.1	Stakeholders identification	3/10/22	14/10/22											
1.2	Interviews finance managers	17/10/22	4/11/22											
1.3	Interviews finance analysts	14/11/22	30/4/23.											
1.4	Interviews administrative staff	3/4/23	15/4/23											
1.5	Scope Definition	17/4/23	21/4/23											
2	Research and bachelor's thesis													
2.1	Definebachelor's thesis objectives	1/3/24	15/3/24											
2.2	Elaborate deeper research on financial markets	16/3/24	15/5/24											
2.3	Case-studies elaboration	15/4/24	25/5/24											
2.4	Thesis delivery and presentation	25/5/24	15/6/24											
3	Redefining the project requirements													
3.1	Requirements integration from thesis and users	15/6/24	30/6/24											
3.2	Functional requirements definition	20/6/24	15/7/24											
3.3	Non-functional requirements definition	1/7/24	15/7/24											
3.4	Workflow definition	15/7/24	20/7/24											
3.5	Data modeling	15/7/24	10/8/24											
3.6	Backend development	15/7/24	10/9/24											
3.7	Frontend development	20/7/24	10/9/24											
4	Project documentation													
4.1	Requirements and workflows	1/7/24	15/8/24											
4.2	Technical development	15/7/24	5/9/24											
4.3	Potential further expansions suggestion	25/8/24	10/9/24											
4.4	Thesis delivery and presentation	5/9/24	13/9/24											

Table 1:Gantt Diagram with time allocation for the different Projects

3.2. Step-by-step Planification

The original project was severely delayed, mostly as a result of the few user interviews. There was inadequate time spent obtaining precise requirements because neither the financial specialists nor the IT consultants prioritized the project. The project's advancement was slow and the overall progress and quality of the solution was affected by this.

The project's basis was established by the initial needs that were collected on-site with the experts. Nevertheless, these first specifications were deficient, as they did not provide the essential information in crucial domains like online asset identification and account administration (which includes differentiating products based on contract or stock codes and following ISO currency standards). Because of this, the solution may not have been entirely successful based alone on these

Further information was discovered via supplementary research that was guided by the business administration thesis. The particularities of managing multiple currencies and utilizing exchange rates in financial transactions were clarified by this study, as was the math behind these operations. The project would have never succeeded without this deeper research because the initial specifications lacked the thorough knowledge required to create a workable solution, especially if it is intended to collaborate with wider range of businesses, each of which has a unique operational style and workflow.

4. Scope of the project

4.1. Project Main Objectives

The primary objective of this project is to develop a web application designed to streamline the registration, management, and reporting of financial operations, including stocks, futures, and options. This tool will be usable by multiple organizations, each with its respective users. The application aims to consolidate and standardize the registry of operations, automate calculations, provide results related to these operations, and generate reports based on the inputs.

In addition to the primary goal, several secondary objectives are essential for the project's success. One key aspect is the integration with external systems. The application will connect to third-party APIs to retrieve relevant market data, including real-time prices, currency exchange rates, and transaction details. This integration will allow the application to interact seamlessly with existing tools used by organizations, such as ERPs or accounting software.

By incorporating live market data, the application can automatically update the value of open positions, calculate results, and provide up-to-date information critical for decision-making.

Even though it will be considered out of scope for this project, building interfaces that allow importing statements from brokerage firms to retrieve operation details and automate the input into the system should be taken into consideration for future upgrades. This functionality will not replace manual inputs, every brokerage firm will have its own standard to export statements, and some might not even allow this. Trading account balance management is also excluded from this project, as it will focus more on the management of operations themselves, the calculations that need to be performed and their implications rather than managing the state of the trading accounts.

4.2. Main Challenges Encountered on the Scope Definition

One of the main issues is that the ability to handle different currencies is also crucial. Since financial markets operate with various currencies, and the exchange rates are constantly varying, the application must support currency exchange rates to ensure accurate conversion and reporting. This feature will allow users to track and manage positions in different currencies, enhancing the application's utility across global markets.

Additionally, the application will offer robust data export capabilities. This functionality is vital for companies that need to migrate data obtained from this solution to other platforms or integrate it with existing systems used in parallel. By allowing data to be easily exported, the application ensures flexibility and interoperability with other business processes and tools.

The user interface should be designed to be intuitive and responsive, accommodating various devices and minimizing the number of user inputs. The goal is to ensure that the application is fast, responsive, and efficient in processing user requests. By focusing on these aspects, the application will enhance the overall user experience, making it easier for users to navigate, manage, and analyze their financial operations.

By addressing these objectives, the project will provide a comprehensive solution that meets the needs of users and organizations involved in managing financial derivatives. The application will not only streamline operations but also offer the connectivity and data management capabilities necessary for effective financial decision-making and reporting.

5. Requirements Gathering

The requirements for this project primarily originate from interviews that were conducted by me during the years 2023 and 2024 with finance staff on a real-world scenario while attempting to find a solution to the situation described in the previous chapter. Moreover, some other requirements that have made the project more extensive and larger, and also applicable to several other companies, come from my bachelor's Thesis in Business Administration and Management, which I wrote and presented in 2024.

During the time I conducted interviews, I also had the chance to observe the outdated, manual workflow firsthand. This allowed me to see where inefficiencies were occurring and identify opportunities for improvement. By analyzing these processes in detail and discussing pain points with the finance staff, I was able to define specific requirements for a tool that could simplify their work. These requirements focused on automating repetitive tasks, improving data accuracy, and integrating seamlessly with the existing ERP system.

The combination of interviews, workflow observations, and requirement definition formed a solid foundation for developing a solution that would directly address the challenges faced by the finance teams while also being adaptable for use by other companies.

5.1. Stakeholder Identification and Analysis

In this project, the primary stakeholders include three distinct types of users, each with different levels of experience and roles within the organization:

Firstly, the main stakeholder and primary user is the finance professional. These users work directly in finance and are actively involved in performing operations in the market. They possess a deep understanding of financial markets and operations, often making complex trading decisions. Due to their expertise, they frequently interact with the system, inputting data, managing operations, and utilizing the system's more advanced features. They are particularly interested in analyzing the outcomes of operations to improve their decision-making and monitor the current financial state.

Finance professionals have significantly influenced the system's design by contributing to the core requirements, including technical specifications, formulas, calculations, the characteristics of different types of operations, market functionalities, and necessary validations. Their insights ensure that the system accurately reflects the complexities of financial operations and supports their analytical needs.

Secondly, in a more operational role, there is the administrative staff. This group includes users with less experience in financial markets, often serving in administrative roles. Their primary responsibility is to input data into the system, typically by transcribing information from broker statements. Unlike finance professionals, these users are not as market-oriented and may require a more guided and simplified interaction with the system. Their feedback is crucial in ensuring that the system remains user-friendly and accessible, even for those with limited financial expertise. For this group, the system has been designed to emphasize straightforward data entry processes, error prevention features, and intuitive workflows that reduce the learning curve.

Finally, at a higher level, the finance director or manager represents another key stakeholder group. While they may not be involved in day-to-day operations or detailed data entry, their focus lies in overseeing the financial outcomes, monitoring performance metrics, and making strategic decisions based on comprehensive analysis. They require the system to provide

accurate, consolidated reports and dashboards that allow them to quickly grasp the financial position and trends without needing to dive deep into operational details. Their feedback has been essential in shaping the system's reporting and visualization features, ensuring that it delivers clear insights and high-level overviews that support strategic decision-making.

These stakeholders have been carefully considered in the design and functionality of the system to ensure it meets the diverse needs of experienced finance professionals, less experienced administrative personnel, and high-level decision makers. By addressing these varied requirements, the system can effectively support both detailed operational work and broader financial analysis.

5.2. Functional Requirements

Functional requirements define the specific behaviors or functions a system must perform. They describe what the system should do, focusing on interactions between users and the system, as well as how the system should respond to various inputs. These requirements are crucial because they serve as the foundation for developing features and functionalities that align with user needs and project goals.

In this context, the functional requirements set the groundwork for implementing the system's operation logs, defining how different operations interact to perform calculations and produce results. Additionally, they ensure that users can easily and quickly access these operations and calculations, minimizing errors and improving overall usability.

5.2.1. User Management

This category covers user registration, authentication and access to different functionalities according to the roles defined, and the multi-organization support and structure of the application:

User Management:

- **Authentication and Registration:** The system allows users to register via an authentication link sent to their email or phone number by their organization. Users belong to organizations, and the organization controls the registration process and decides which users should be allowed into the application.

- **Access Control:** Access to sensitive data is restricted based on user roles and organization affiliation. To begin with, at least three types of users will be accessing the system. Firstly, a user with total control over data and functionality, who should be able to sign up other users for that organization. Secondly, the regular user who will be operating and registering operations. Finally, a user with only viewing permissions is defined, but cannot interact with the system. Evidently, only users belonging to the organization that owns the information can view and manipulate it.

Multi-Organization Support:

- **Core functionality:** The platform should support multiple organizations, ensuring that each organization’s data is isolated and only accessible by its own members. The system should allow the existence of one or more people who would be in charge of defining the users from that organization, meaning adding, removing and managing their roles within the system.

- Role-based access control allows different permissions and levels of functionality based on the user's role within the organization (e.g., finance professionals, administrative staff, finance directors).

5.2.2. Settings and Parametrization

- **Core functionality:** The system should allow certain users that the organizations consider more experienced, to be able to define and perform parametrization within the system.

- These expert users are responsible for setting up critical parameters such as defining the Markets and Trading Accounts with their respective currencies, which are essential for the system's operations. Additionally, they manage administrative tasks like adding new users to the platform and configuring their access levels. This ensures that the key settings are accurately configured and aligned with organizational needs and keeping alignment with the organizational hierarchy of the organization itself.

5.2.3. Operations Management

This category focuses on the core functionality of managing financial operations within the application, including the creation, consultation, editing, and deletion of operations, as well as the inputs needed to accurately define each operation within the system.

This involves capturing the key data required for the classification and management of operations of different types, ensuring that relevant information such as the operation type, associated entities, and transaction specifics are consistently documented, so that the system can extract this information and perform the necessary calculations later. Moreover, the system should support the proper tracking, analysis, and control of financial operations across the different transaction types.

Operation Creation:

- **Core functionality:** Users can create various types of operations (e.g., stock, futures, options) with detailed information and specific fields related to each operation type. OTC (over the counter) operations are left out of the scope of the project due to its complexity, customization possibilities and lack of standardization.

- The functional requirements for the operations in the system encompass common characteristics shared by all types of operations, while also addressing the specific attributes that vary according to the type of operation being performed.

- All operations have a unique ID, a date, a linked market, a trader responsible for the transaction, and a description. Each operation should belong to a unique operation chain and must be associated with a specific organization. Operations can be classified into three types: stock, futures, and options. The type of operation is determined by the specific class it belongs to, and this type cannot be edited once set. Additionally, operations can be locked to prevent further modifications, because altering operations after results have been triggered would cause inconsistencies and users should not be

allowed to do that. User roles such as creators and modifiers are tracked for each operation, ensuring a clear audit trail[1].

- Depending on the type of operation, certain attributes or characteristics vary from each other:
    - **Stock operations:** These include the stock code, the number of shares involved, and the price per share the operation was conducted on.
    - **Futures operations:** These include the contract name and price per contract.
    - **Options Operations**: These include the strike price, premium, and price per contract and if it is the case of a call or a put option.
- All operations may involve commissions, the requirements for which will be defined separately.
- The system ensures that operations are created with all these necessary data and should warn the user if one or more inputs are missing.

Operation Editing:

- **Core functionality:** Users must be allowed to update existing operations, to solve potential mistakes during manual input of operations. However, once an operation is locked (e.g., results have been calculated or exported), it cannot be updated further. Operations that are locked or part of a finalized chain cannot be edited to prevent inconsistencies.

Operation Deletion:

- **Core functionality**: Users should be able to delete operations from their organization. The system that only unlocked operations can be deleted.

Commission Management:

- **Core functionality**: Users can create, view, and delete commissions, which can be linked to operations. Multiple types of commissions, which should be able to be defined by users from the different organizations that use the application can be applied to an operation simultaneously. Commissions can be added or deleted if the operation that they are being defined into does not belong to a closed chain of operations, as it would alter calculations and results.

[1] While auditing is not part of this project, tracking details of who created or modified certain information are included to ensure the system's structure supports future expansion if audit tracking becomes necessary.

5.2.4. Operation Chain Management and Calculation

Operation chains are designed to logically group related operations, allowing them to interact with each other or be treated as interconnected for specific financial calculations. This ensures that operations which are part of the same chain share consistent attributes and are processed together when determining results, given some prerequisites and validations are conducted when dealing with them. This group of functional requirements addresses the handling of operation chains, and the calculations involved between them.

Operation Chain Creation:

- **Core functionality**: Users can create chains to group related operations of the same type (e.g., stock operations). Chains are designed to handle multiple operations while ensuring homogeneity (no mixing of operation types within a chain). The system only allows for the creation of an operation chain if at least an operation is linked to that new group. Operation chains that do not contain any operations are meaningless and should not be allowed to be created.

Operation Chain Editing:

- **Core functionality**: Users should be able to link or unlink operations to and from an existing chain, provided the chain is not locked or closed, because if an operation is locked it means the results have been validated and should remain unaltered. The system ensures that operations within a chain remain consistent, and that no operation is part of multiple chains at the same time. Or that the operations that make up the chain are not altered when the chain is closed.

Operation Chain Deletion:

- **Core functionality**: Users can delete operation chains, but only if they are not locked or closed. When a chain is deleted, all associated operations must either be deleted or unlinked prior to deletion.

Chain Integrity and Validations:

- **Core functionality**: The system automatically validates and maintains the integrity of operation chains, ensuring that no conflicting operations are linked within the same chain. Once a chain is locked or finalized, no further changes are allowed to either the operations within the chain or the number of operations linked to it. This ensures that the state of the operation chain and its components remains unaltered once the results are generated and potentially exported to another system.

5.2.5. Markets, Trading Companies, and Trading Accounts Management

This category covers the entities essential for linking operations to real-world market data and ensuring that trades are accurately tracked across different currencies, markets, and accounts.

Market Management:

- **Core functionality**: Users should be able to create and delete markets, specifying details like market name, the market identifier, which is unique, currency, trading days and a short description. The system will ensure that there cannot be two markets with the same market identifier, which is unique, and does would allow the deletion of markets linked to active operations, given that it would leave operations incomplete and disable the data market retrieval functionality.

Trading Company and Trading Account Management:

- **Core functionality:** Users should be able to create and delete trading companies and their associated accounts. Each trading account is linked to a specific trading company and contains key attributes like an account number and currency. It should not be possible to have two trading accounts with the same account number. Operations must be associated with a trading account to ensure accurate tracking and reporting, and also retrieving which currency the results need to be converted into.

Relationship Between Markets, Trading Accounts, and Trading Companies:

The system should accommodate the complexity of real-world trading scenarios by allowing flexible relationships between markets, trading accounts, and currencies. A market typically operates in a specific currency, while trading accounts may exist in a different currency. This setup would allow for scenarios where a user trades in one currency while operating in a market that uses another. For example, a user might use a trading account in EUR to operate in the NASDAQ market, which uses U.S. dollars. This level of granularity would allow the system to handle operations across different markets, accounts, currencies, and traders, ensuring accurate conversions, reporting, and results.

This approach also ensures the application would be functional with future expansions of the application with updates such as account balance management.

5.2.6. Market Data Access and Provisional Profit and Loss Visualization

The system must access and retrieve real-time market data to provide users with the current value of ongoing (chains of operations that are defined as not closed) chains of operations. As market conditions fluctuate daily, the system will calculate a provisional Profit/Loss (P/L) for each operation chain. This provisional P/L reflects unrealized gains or losses, offering insights into how the portfolio is valued at a specific point in time.

Not only it has to be noted that the value of these assets fluctuates constantly, the exchange rates between different currencies also vary similarly. Users should have access to the different quotes that are relevant to them and the system should ensure values that are consistent with the exchange rate of the time of consultation and remain updated for future consultations.

Provisional P/L Calculation:

- **Core functionality:** Profit and Loss (P/L) metrics are crucial for assessing trade and investment performance. Profit indicates financial gain, while loss denotes a decrease in value. Accurate and real-time provisional P/L calculations are essential for effective portfolio management, as they allow users to monitor potential gains or losses dynamically. This feature helps financial managers make informed decisions and adjust strategies promptly without having to wait for trades to close, providing a comprehensive view of the portfolio's performance.

- Additionally, the system should detect divergence within market currencies and the trading account or commissions currencies and should require for a user exchange rate to convert the results to a currency that matches the one of the trading accounts. In the case that currency coincides, no further inputs are required, and the system should consider the values that already exist in that trading account currency.

    - The user should get an approximate value on the open position, therefore, from the markets the end-of-day quotes shall be the ones employed to value the positions. This approach can be used for stocks, for the contracts of the futures and options and should be used for the currency exchange rates.

  For operation chains that are still open (meaning that some trades are pending), the system should calculate the provisional P/L based on:

    - The partial results of completed operations (e.g., purchases and sales).

    - The current market value of any open position is still active and changing.

  Users can visualize the provisional Profit/Loss (P/L) in real-time as the market value updates daily. This provides insight into how each position is valued before it is closed, offering a snapshot of the current financial status of ongoing trades. While these values are typically accessible through brokers' applications, integrating this functionality within the system ensures that management can efficiently view both the results of completed operations and the provisional results of ongoing trades without needing to manually access each broker’s platform. This consolidated view streamlines portfolio management and helps decision-making by providing a comprehensive and up-to-date overview of all trading activities carried out within through many different traders.

Final P/L Upon Chain Closure:

- **Core functionality:** Once an operation chain is closed and locked, the provisional P/L becomes definitive. At this stage, no further updates to the operation chain or its components are allowed.

- The final P/L is calculated and locked in, ready for export to other systems or for reporting purposes. These values are considered final and cannot be altered, ensuring consistency and accuracy across the system.

5.2.7. Operation Chain Closure and P/L Calculation

The system should provide functionality for users to close an operation chain when preconditions are met. Closing a chain triggers the final P/L calculations, ensuring that all operations within the chain are properly accounted for according to their specific characteristics. To carry out an operations chain closure, the system must ensure some conditions are met.

Chain Closure Preconditions:

- Users can initiate the closure of a chain if all relevant operations are complete, and no outstanding actions remain (e.g., all trades have been executed, and no pending). This can be verified by checking the number of shares purchased and sold or contracts. If the number is unequal, the chain should not be allowed to be closed.

- As validations, the system should verify that all previous validations are carried out and that the present data is complete in order continue with the closure and locking, ensuring the integrity of the results. This is a crucial step because once the chain is closed, it is considered that the operations are settled and therefore its results can be exported elsewhere (accounting software, reports, sent to management, etc.), therefore, wrongful statements could implicate severe consequences.

Realized Profit/Loss Calculations:

- **Core functionality:** Once the chain is closed, the system performs calculations to determine the final P/L, with distinct methods depending on the operation type.
    - **Stocks:** P/L is calculated based on the difference between purchase and sale prices of the shares, accounting for commissions.
    - **Futures:** P/L is determined based on the settlement prices of the futures contracts and the direction of the trade (long/short).
    - **Options:** P/L is calculated considering the premium, strike price, and the value of the underlying asset, incorporating complex option-specific rules.
- Similar to the provisional Profit/Loss calculation, the system should detect divergence within market currencies and the trading account or commissions currencies and should use the values that have been converted to the trading account's currency, so that all operations within the chain and the commissions match the trading account's currency. Otherwise, the result would be a mix of values of different currencies which cannot be used to make calculations together. The currency exchange rate should be defined by

the user itself and it cannot be found anywhere else other than in the broker' statements, as these exchange rates fluctuate all the time.

Finalization and Locking:

- **Core functionality:** After the calculations are complete, the system should lock the operation chain, preventing any further modifications. The results are stored as definitive and can be exported or used for reporting. Once locked, the chain's data is immutable, preserving the integrity of financial records and ensuring that the P/L values are consistent across the system.

- Because of that, the system should not allow changes in any of the operations pertaining to that chain either, effectively locking all data that depends on that group of operations.

5.2.8. Data visualization, Analysis and Reporting

This group focuses on features designed to support financial analysis and strategic decision-making:

Operation Dashboard:

- **Core functionality:** Users should be able to access a centralized summary page within the operations menu that provides an overview of all individual operations. This summary page should display the main attributes of each operation, such as the operation type, date, market, trading account, stock code or the contract, and the prices. Users should also be able to view the current market value of each open position. Additionally, there should be an option to access further details for each operation if desired, allowing users to review more in-depth information about specific transactions which would not fit the main dashboard.

Operation Chain Dashboard:

- **Core functionality:** Users can analyze aggregated results at the chain level, including key performance metrics such as gross profit/loss, total spending on commissions, and overall operational efficiency. The system presents these indicators by breaking down the results into gross profit, net profit, and commissions. This comprehensive breakdown provides a detailed view of how interconnected operations contribute to the final financial results and helps users understand the impact of commissions on overall performance.

Tables, Filtering, and Search Capabilities:

- **Core functionality:** Users should easily find specific information or drill down into subsets of operations to access and download information quickly. That includes, searching and finding operations or chains of operations.

- Interactive tables summarize operations and chains, with features for filtering, sorting, and searching based on various criteria such as date, type, market, stock/contract, or trading account.

- The user should be able to download the information that they are visualizing in a format that can be read by the common typical spreadsheet editors like Microsoft Excel or Google Sheets. This functionality should exist for the operations and the chains of operations dashboards, where the main information and results are provided. A feature that covers this requirement would facilitate further analysis outside the application and readability on widely used tools that users are used to employ to handle and manipulate data.

Detailed Performance Analysis:

- **Core functionality:** The system should allow the user to have the capabilities and make information available so that the user can analyze performance at deeper levels. For example, by individual stocks, contracts, markets, traders, etc. This offers deep insights that go beyond overall chain performance.

- This granularity helps users see exactly which assets or categories are driving profits or losses, allowing them to make more informed decisions. By comparing different types of trades or financial instruments, users can identify trends, spot opportunities, and optimize strategies for better returns. At this level of detail, it would be easier to evaluate in which stocks or contracts the company is underperforming, where commission costs are higher than usual, and other similar factors. These insights help determine which traders, brokers, or products are most effective. Overall, these metrics give management a clearer understanding of where the company is falling short, enabling them to take action to improve efficiency on trades.

5.3. Non-Functional Requirements

5.3.1. Performance

The application should load and render pages, including those that contain lists of possibly hundredths of operations within two seconds under typical network conditions.

The system should be designed to handle a significant number of concurrent users without noticeable performance issues. While the exact number of users is not specified, the system should be easily scalable to accommodate increased demand. It must maintain performance even under peak loads, where multiple users may be generating reports or interacting with various dashboards.

5.3.2. Scalability

The platform should be set up in a way that could handle the growth of users and companies joining over time by expanding both the frontend and backend components to keep up with the increasing demand and ensure good performance and reliability.

Expanding horizontally means incorporating server instances to manage the growing workload instead of just upgrading a single server unit alone—a method that enables the system to effectively handle greater numbers of concurrent users and transactions while also improving fault tolerance through load distribution; if one instance encounters issues and fails to function as intended; others can step in to keep operations running smoothly and minimize downtime.

Moreover, horizontal scaling offers adaptability and scalability by allowing the system to automatically allocate resources according to requirements. This guarantees performance and cost effectiveness as the user base grows and peak workloads vary, thus readying the platform, for future expansion without requiring extensive restructuring.

5.3.3. Responsiveness and Reactivity

When it comes to the UI (User Interface), all screen sizes should be completely supported by the application, guaranteeing usability on desktop and mobile browsers. When processing large amounts of dynamic data, the application should nevertheless maintain seamless interactions, real-time reactivity, and quick reaction times.

In order to give consumers an effective and pleasant experience across all of the supported devices, the solution’s responsiveness is an essential component since it guarantees that users may access and utilize the platform with ease on a variety of devices, such as PCs, tablets, and smartphones. Users want a consistent and optimal experience across all devices in today's mobile-first environment. The application can enhance user satisfaction and engagement by adapting its layout and functionality to multiple screen sizes and orientations by implementing responsiveness. This adaptability enhances performance and usability, increases accessibility for a wider audience, and ultimately boosts user retention and the application's success.

The reactivity should guarantee that any updates to data or state are quickly reflected across the interface without delays or without manual refreshes. This includes fast updates to the information display as well as instantaneous reaction to user actions like clicks and data inputs. Reactivity ensures that the system responds to modifications and user inputs, which is essential for preserving a fluid user experience.

5.3.3.1. Essential Features of a Reactive Design

- **Immediate Feedback:** Users should promptly see the results of their actions, whether these actions involve keystrokes, mouse clicks, menu selections, or other types of input.
- **Clear Data Status:** Users should always be aware of the current state of their data. For instance, they should easily confirm if their changes have been saved or if a backup has

been overwritten. In applications like drawing programs, users should see whether a line segment consists of smaller segments.

- **Accessible Help:** Users should always know how to access help, which should be substantial and relevant. Effective help may be context-sensitive or modal, but it should provide clear explanations rather than just a collection of screenshots or menu labels.

5.3.4. Security

The system should implement robust authentication and authorization practices to securely manage user sessions. It must ensure that all sensitive data is encrypted both in transit using HTTPS and at rest where applicable. Comprehensive protection against common web vulnerabilities is essential, including defenses against XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), and SQL Injection.

In addition, the system should incorporate input validation and sanitization to prevent malicious data from being processed. Secure communication protocols must be used to protect data transmission, and error handling should be designed to prevent sensitive information from being exposed to users.

It is not within the scope of this project, but it would be recommendable that regular security audits and vulnerability assessments are done to identify and address potential weaknesses proactively, meaning for which they are specifically searched.

Fortunately, the research needed for this project into different technologies that might be used in this project show that many modern backend frameworks come with built-in features and tools that streamline the implementation of security measures. This makes it easier to enforce strong protection and maintain the integrity of the system.

For example, authentication management, session control, and CSRF protection are a few of the crucial built-in technologies that are available in a variety of contemporary frameworks.

Sensitive information should protect via authentication, which would then limit access to authorized individuals. In order to prevent attacks like session fixation, session ID management keeps track of user behavior and upholds session security. By preventing cross-site request forgery, CSRF tokens guarantee that user actions are carried out knowingly. When combined, these methods aid in securing user interactions and shielding the program from frequent flaws.

5.3.5. Reliability

The app should have an uptime of at least 99.9% during production, with mechanisms in place for error handling and recovery. Utilizing cloud providers can significantly contribute to achieving this high level of reliability, if defined in a Service-Level Agreement[2].

Cloud platforms often offer robust infrastructure and services designed to enhance uptime, including automated load balancing, redundant systems, and geographically distributed data centers.

The system should be capable of managing unexpected failures or crashes effectively, ensuring that it recovers quickly and without data loss. This includes implementing error handling and backup procedures to preserve data integrity and maintain operational continuity during disruptions.

5.3.6. Maintainability

The codebase should be modular, following the principles of separation of concerns, to facilitate easy updates and maintenance, and the platform should include comprehensive documentation and adhere to coding standards to ensure that new developers can easily onboard and contribute.

5.3.7. Compatibility

The app should be compatible with all major browsers (Chrome, Firefox, Safari, Edge) and their mobile versions. The system should reflect a consistent and functional user experience across these platforms.

While the primary focus is on supporting the most recent versions of these browsers, it is also important that the application remains functional on slightly older versions to accommodate users who may not have the latest browser updates. However, critical functionality should not be compromised by the need to support outdated versions. The goal is to prioritize a seamless and effective user experience while maintaining broad accessibility across modern and commonly used browser versions.

5.3.8. Usability

[2] A Service-Level Agreement (SLA) is a formal contract between a service provider and its customers that specifies the services to be provided and the standards of performance the provider must meet. It details metrics such as uptime, response times, and support availability, setting clear expectations for service delivery and performance. In contrast, a Service-Level Commitment (SLC) is a more generalized form of an SLA that outlines broader service expectations without the same level of detail. While an SLA provides specific, measurable criteria and penalties for non-compliance, an SLC offers a more flexible approach to defining service standards and commitments. Both are essential for establishing the framework for service quality and accountability between providers and their customers.

The UI should be intuitive and user-friendly, designed to facilitate ease of use with clear and consistent navigation throughout the application. Menus, buttons, and workflows should be logically organized and easily discoverable, minimizing the learning curve for new users. Visual cues, such as tooltips and contextual help, should be provided to guide users through complex functions and features.

Additionally, the system must incorporate accessibility features to ensure it is usable by individuals with disabilities. This includes adherence to ARIA (Accessible Rich Internet Applications) standards to enhance the compatibility of interactive elements with screen readers and other assistive technologies.

The application should support keyboard navigation, allowing users to perform all actions without relying on a mouse. Color contrast should be sufficient to accommodate users with visual impairments, and alternative text should be provided for all images and media. These features collectively ensure that the app is inclusive and accessible to a diverse user base.

5.3.9. Data Integrity:

The app should ensure consistent and accurate data handling, especially during operations like creation, updates, and deletion.

Transaction management should ensure that all database operations are either fully completed or rolled back in case of failure. This is particularly crucial during processes such as data exports or complex calculations, where partial or incomplete operations could result in inconsistent or incorrect data. Such inaccuracies could lead to erroneous data being used in external systems, reports, or financial analyses, potentially causing significant issues downstream.

Ensuring robust transaction management helps maintain data integrity and reliability, thereby preventing the propagation of errors and safeguarding the accuracy of critical operations.

5.3.10. Localization and Internationalization:

The system should support multiple languages and be easy to adapt to different regions, currencies, and date formats if required in future phases of development. Although the initial requirements and development will be conducted in English, it is worth noting that software does not remain immobile and changes overtime to adapt to new requirements or new customers that may become interested.

Therefore, incorporating localization and internationalization features from the outset can be a valuable advantage for future enhancements. This foresight ensures that adding support for additional languages and regional settings will be more straightforward and cost-effective as the application evolves, potentially broadening its market reach and usability.

5.3.11. Data Portability

The app should support exporting data in multiple formats (e.g., CSV, Excel, JSON, XML) to ensure compatibility with various external systems. The intended formats should include those commonly used and widely accepted by standard software applications to facilitate smooth integration with ERP systems, accounting software, and other reporting tools. The exported files should be properly formatted and include essential metadata, such as timestamps and identifiers, to ensure that data is easily interpreted and integrated by external systems, enhancing the overall interoperability and usability of the application.

5.3.12. Integration Capability

The app should offer API endpoints or webhooks to allow automated data retrieval by external systems. This ensures that market quotes can be pulled directly into third-party software without manual intervention.

The system should support batch exports to handle large volumes of historical data when necessary.

5.3.13. Data Accuracy and Consistency

The daily retrieval process should ensure that market quotes are accurate, up-to-date, and free from inconsistencies.

Any discrepancies in the data should be flagged and logged, ensuring clean data is exported to external systems.

5.3.14. Automation and Scheduling

The daily retrieval process must ensure that market quotes are accurate, up-to-date, and free from inconsistencies. Any discrepancies in the data should be flagged and logged, ensuring that only clean data is exported to external systems.

Additionally, the system should notify users immediately if any errors or issues occur during data retrieval or processing. This transparency helps prevent users from placing undue trust in potentially erroneous information, allowing them to verify and address any issues before making decisions based on the data.

5.3.15. Data Volume Handling

The app should efficiently handle large datasets of daily quotes and export them without performance bottlenecks. Given that the volume of data is indefinite and can grow significantly, the system must be capable of managing extensive historical data and large quantities of securities. Additionally, the application must accommodate the substantial amount of

information retrieved from online market APIs, which may require making hundreds or even thousands of queries concurrently to meet the demands of multiple organizations. Ensuring optimal performance and scalability in these scenarios is crucial to maintaining responsiveness and data integrity as the system evolves.

5.3.16. Data Retention and Archiving

The system should establish clear retention policies for storing and archiving historical quotes, ensuring that data is preserved for long-term analysis or re-export as needed. As the application scales and the volume of data grows, it will be beneficial to incorporate advanced data management solutions. In future expansions, implementing a data lake or a specialized database optimized for handling large-scale data can enhance the efficiency of storing and managing extensive datasets.

Data lakes are designed to accommodate vast amounts of structured and unstructured data, offering scalable storage solutions and facilitating complex data analysis. This approach could not only support the long-term storage of logs and market data history but would also improve the system's ability to handle high data volumes and maintain performance over time, especially as the platform gets more users aboard.

5.3.17. Auditability

The system should maintain detailed logs of all export activities, including timestamps, formats used, and destination systems. This practice is essential for several reasons:

- **Traceability and Accountability:** Logging export activities ensures that each data transfer is recorded and can be traced back if issues appear. This traceability helps in identifying and resolving any discrepancies or errors that may occur. By having the traceability, we ensure that we are providing a clear audit trail that supports transparency and accountability.

- **Data Integrity:** Detailed logs help verify that data exports are completed accurately and received correctly by the intended destination. If problems occur with the data at the receiving end, logs can be reviewed to determine if the issue originated during the export process or afterward, thereby protecting the integrity of the data.

- **Troubleshooting and Support:** Logs provide valuable information for diagnosing and resolving issues related to data exports. By capturing details about what was exported, when, and where, support teams can more efficiently address and correct any problems, minimizing downtime and ensuring smooth operation.

- **Auditability:** Maintaining comprehensive export logs supports internal audits and reviews by providing a historical record of all export activities. This capability is crucial

for tracking changes, verifying processes, and ensuring that operations are conducted as intended.

Overall, proper logging of export activities enhances the reliability and efficiency of the system, offering users a transparent and accountable way to manage their data transfers.

6. Workflows Definition

Understanding and reflecting the workflows of users is crucial for defining an effective solution in this project. The workflows map out how users interact with the system, from logging in to performing specific operations like creating stock, futures, or options market operations.

By visualizing these processes, we can identify key decision points, dependencies, and potential inefficiencies. Therefore, by giving a global look into the processes carried out by users prior to the implementation and deployment of this solution in a real work environment ensures that the system will be designed to accommodate users' needs intuitively, streamlining their actions while minimizing errors and confusion.

Additionally, clear workflows help align the development efforts with actual user behavior, resulting in a solution will be both practical and user-friendly.

6.1. User Workflow Overview

For this project, it has been decided to make a flowchart that summarizes the user's interaction with the operation based on the requirements. The decision to use an operational flowchart to define the workflow is rooted in the clarity and structure it provides. The main advantage of using flowcharts in software engineering is that they visually represent the sequence of actions and decisions in a process, making it easier to understand the user's journey through the system.

This approach offers several advantages, such as a clear and straightforward visual representation of complex workflows, which makes it easy to see the different steps involved in the workflow and understand how they are connected to each other, aiding in detecting dependencies that are going to exist.

Flowcharts also help detect key steps by laying out the process step-by-step, they help identify critical actions and decision points, ensuring that no part of the workflow is overlooked. This fact has a relation with readability, since a workflow can provide a visual representation that is very clear and straight to the point summarizing all steps in a small format that allows the development team and the stakeholders to identify the steps involved. Visualizing the workflow in a flowchart format makes it easier to spot potential bottlenecks, redundancies, or errors in the process.

It also improves communication because flowcharts serve as a universal language that can be easily understood by all stakeholders, from developers to non-technical team members, facilitating better communication and collaboration, while written requirements are not as simple to comprehend.

Therefore, well-defined flowcharts help in planning and implementing the system by providing a clear roadmap, ensuring that the development remains aligned with user needs and project goals. This information is easily comparable with user input and helps when contrasting it with user feedback.

The following flowchart on Figure 1 intends to represent most of these points for this project.

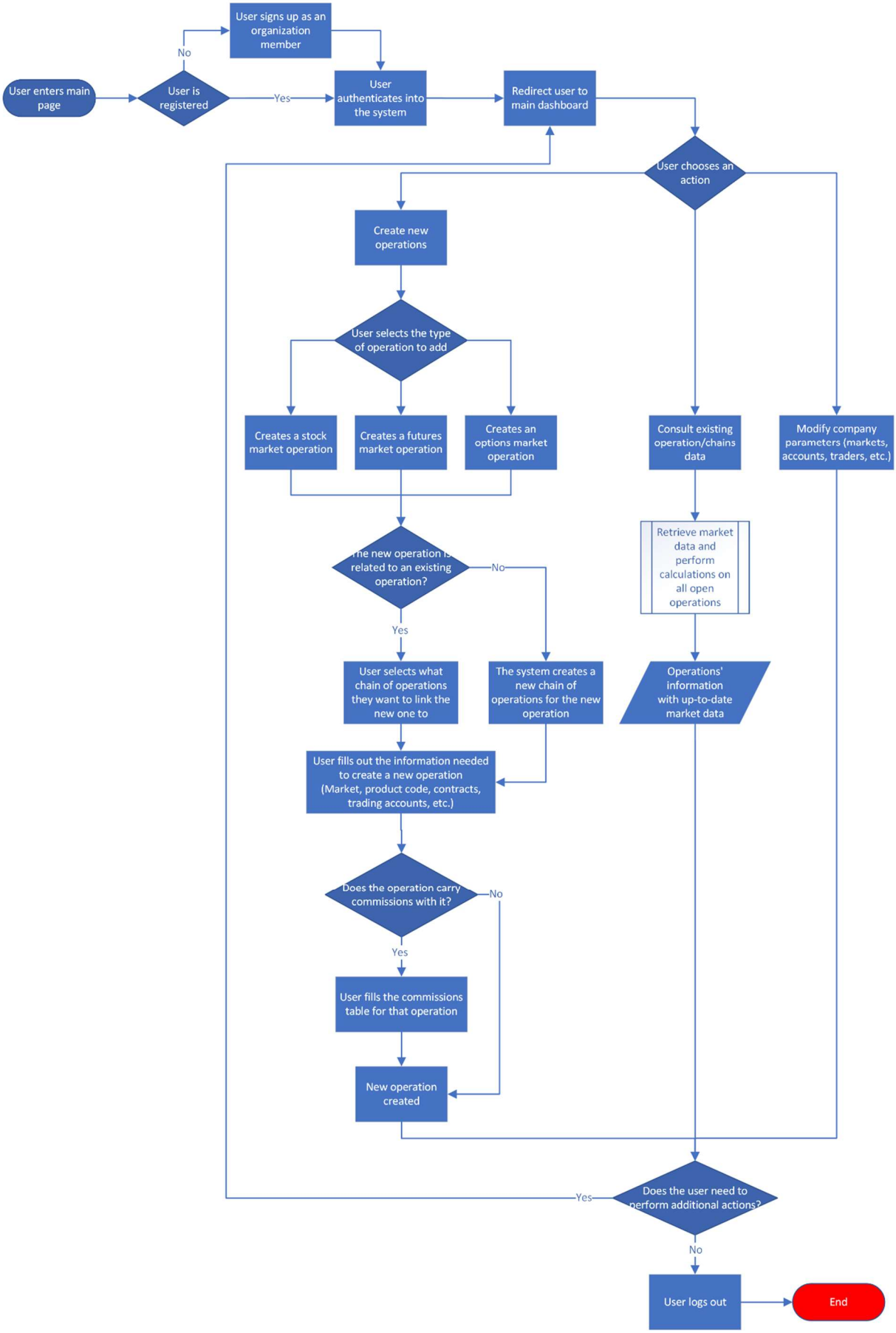

Figure 1: User operations workflow

6.2. Use Cases Overview

A clear way to represent the use cases that arise from the system's requirements is through a UML use case diagram. This approach is a representation of the interactions between users (actors) and the system, illustrating how different functions and features are accessed. By mapping these scenarios in a use case diagram, it becomes easier to understand the relationships between user roles and the corresponding actions they perform within the system, providing an organized representation of the system's functionality.

6.2.1. Main Advantages of Using a Use-Case Diagram

A use-case UML diagram helps to summarize and compress information by illustrating use cases and their relationships with actors. Some of the main advantages for this project include:

- **System Design Improved Clarity:** One of the main advantages of a use-case UML diagram is that it provides a clear and organized visualization of the system's functionalities and the roles of different actors. This clarity is crucial for both developers and stakeholders, as it simplifies the understanding of system requirements and the interactions between users and the system.

- **Identification of User Roles and Permissions:** This type of diagram assigns different actors to the use-cases. By mapping out the various actors and their associated use cases, the diagram helps identify the necessary roles within the system. This, in turn, facilitates the definition of user permissions and access control, ensuring that each actor has appropriate access to system features.

- **Facilitates Communication Among Stakeholders:** The use-case diagram acts as a common language among developers, analysts, and non-technical stakeholders. It enables all parties to have a shared understanding of the system's functionality, which is essential for effective collaboration throughout the iterations of the development process.

- **Support for System Validation:** With the use-case diagram, stakeholders can validate the completeness and accuracy of the system requirements. Each use case represents a specific requirement or functionality, allowing stakeholders to verify that all necessary operations are accounted for and correctly implemented.

- **Foundation for Further System Development:** The diagram provides a foundation for further system development, such as detailed system modeling, sequence diagrams, and activity diagrams. It serves as an initial step in the transition from high-level requirements to detailed design and implementation.

6.2.2. Use Case Diagram for the Financial Operations Management System

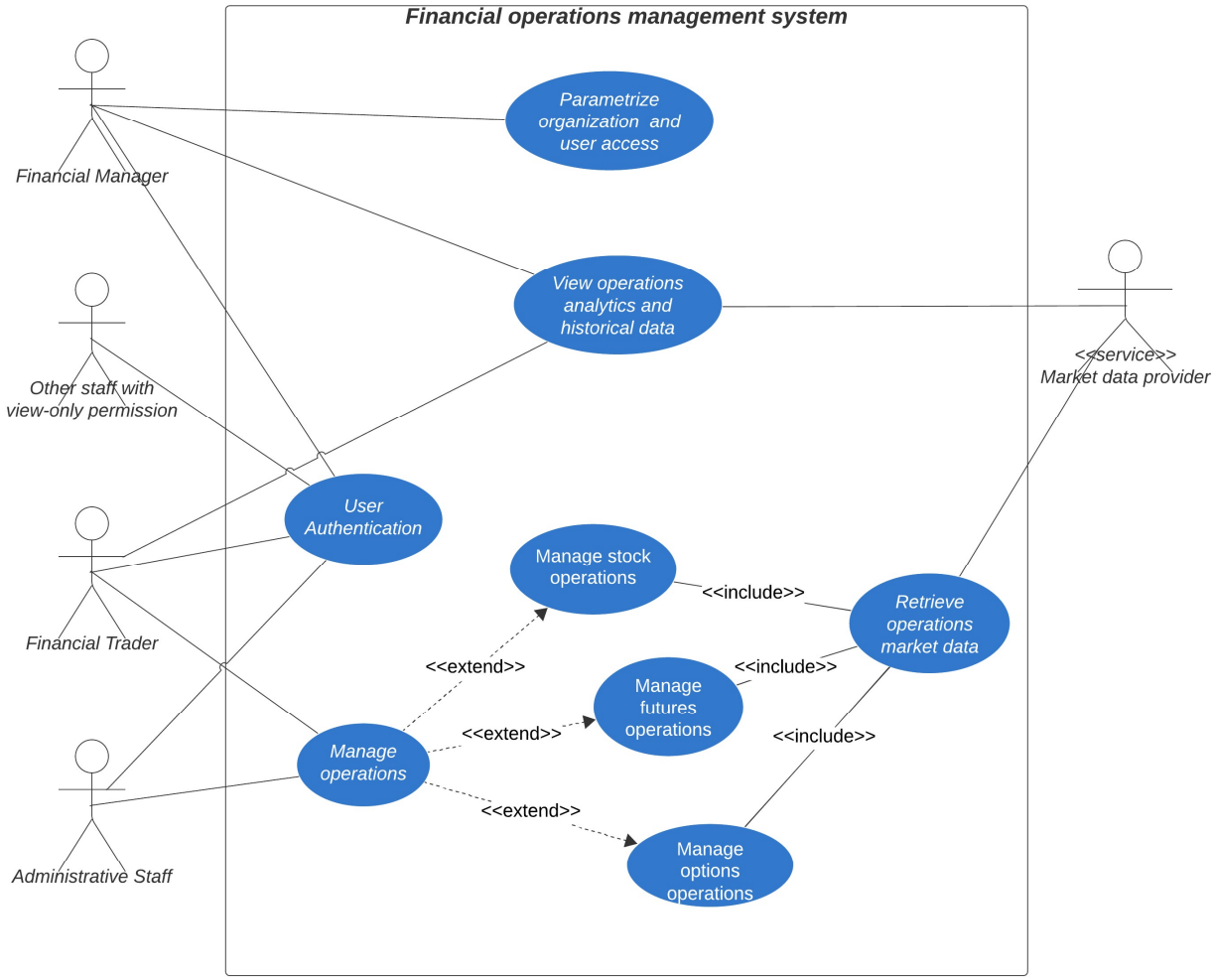

Figure 2: Use-case UML diagram

A use-case diagram like the one shown in Figure 2, serves as a foundational tool in the analysis and design of the system, visually outlining the interaction between users (actors, on the left) and the use-cases that will be solved by functionalities of the system and other external tools or services.

In the context of the Financial Operations Management System that this project intends to develop, the use-case diagram illustrates a comprehensive view of how various stakeholders interact with different components of the system.

6.2.3. Key Elements of the Use-Case Diagram

6.2.3.1. Actors

The actors that appear in the use-case diagram are the following:

- **Financial Manager:** This actor plays a pivotal role in the system, with access to critical functionalities such as parametrizing organization and user access and roles. Apart from the fundamental settings, it is also in charge of managing operations, and has an interest in viewing analytics and historical data. The financial manager's implication in many different use cases represents their responsibility for overseeing and configuring the system.

- **Financial Trader:** The Financial Trader is focused on the core financial activities, specifically managing stock, futures, and options operations. This actor interacts directly with the system to execute and monitor financial transactions.

- **Administrative Staff:** This actor's role is primarily supportive, managing operations that might not require the specialized financial expertise of a trader but still involve interaction with the system's operational aspects.

- **Other Staff with View-Only Permission:** This actor is granted limited access, allowing them to view operations analytics and historical data without the ability to modify or interact with other system functions. This role is essential for ensuring that relevant stakeholders can access necessary information while maintaining system integrity.

- **Market Data Provider:** As an external service actor, the Market Data Provider supplies crucial market data that the system retrieves to enable informed decision-making by the internal actors.

6.2.3.2. High-Level Functions Involved

The actions or functionalities which users carry out according to the use-case diagram are:

- **Parametrize Organization and User Access:** This use case, exclusive to the Financial Manager, allows the configuration of system access and organizational settings, ensuring that the right users have the appropriate permissions, and that the system aligns with the organization's needs.

- **View Operations Analytics and Historical Data:** This use case is shared among all internal actors but with varying degrees of access. It provides valuable insights into past and present operations, which are critical for informed decision-making and strategic planning.

- **User Authentication:** A fundamental use case that ensures all users are verified before accessing the system, maintaining security and data integrity.

- **Manage Operations:** Central to the system, this use case is extended by more specialized cases like managing stock, futures, and options operations, which are core to the financial trading process.

- **Retrieve Operations Market Data:** This use case involves interaction with the Market Data Provider, ensuring that up-to-date market information is available for all financial operations within the system.

6.2.4. Interaction with Internal and External Actors

The interaction between internal and external actors is key to the functionality the platform this project intends to develop, since an important part of the data necessary to obtain results based on the user inputs is only accessible via third party software solutions. We can obtain types of interactions based on the diagram:

- Internal actors such as the financial manager and financial trader interact with the system to perform tasks directly related to their roles, such as managing financial operations and taking decisions based on analytics that the system will output for them. The diagram highlights these interactions, showing how the system supports the workflows and responsibilities of each internal actor.

- External actors, such as market data providers, have the primary function of first retrieving the necessary quotes or market data from the system. They then return this data to the system, allowing it to perform the required calculations. Obtaining all the necessary quotes is crucial for the system's correct operation. The interaction with this external service is essential for acquiring real-time market data, which feeds into the system's operations.

6.3. Conclusion

The process of understanding and defining the different workflows, use-cases and interactions that involve internal and external actors is crucial for understanding system requirements and planning the development phase of the solution. Mapping user interactions, decision points and dependencies help identify critical areas. The diagrams also clearly help in identifying the role of external actors that will interact with the system and with the user, which has been the case in this project.

They have helped identify different user-roles, permissions, and all necessary functionalities. The communication between the developers and the stakeholders is also much easier when using these sorts of diagrams, as requirements can become very heavy and complicated to understand at a certain point.

In summary, incorporating these visualizations into the design improves the planning, synthesizes the requirements gathered from all the stakeholders and integrates them in understandable diagrams that include all the information from a high-level standpoint.

7. Solution Architecture: An Effective Design and Structure for the System

It takes a well-defined architecture to create a reliable solution. Every software solution needs a strong architecture that not only enables the platform to be developed quickly and dependable but also makes it easier to update and change in subsequent project iterations when new user requirements are identified.

The project's functional and non-functional criteria ought to be met by this design. Determining interactions between various layers (such as frontend, backend, and database) and guaranteeing scalability, maintainability, and performance are some of the crucial aspects in the process.

The primary technologies that have been involved in creating this application and taking all of these requirements into account are meant to be explained in this chapter, together with an explanation on the approaches taken to solve the main challenges that have been encountered during the development phase.

7.1. Data Models and Structures

One foundational aspect of the architecture of the proposed solution is the data modeling. Data modeling involves structuring and organizing data to meet the application’s needs, ensuring efficient storage, retrieval, and analysis.

Visual representations of the relationships between the different entities that the project includes can be shown through UML class diagrams. A UML class diagram can help clarify relationships, dependencies, and data flows, aiding in the design and optimization of database schemas and API interactions.

Not only a UML class diagram is helpful to get a fast, simple look on the project data interactions, it also helps in defining the models for the backend during the development stage, ensuring that the system's data handling is well-organized and efficient, without redundancies or wasted resources. Therefore, it was deemed necessary for this project to set up a UML class diagram.

The following UML class diagram in Figure 3 represents the entire schema of the proposed solution. It illustrates the database classes, their attributes, and the relationships among them along with some of the functions needed on the backend side.

This proposed solution focuses on a modular design that ensures flexibility and scalability. This architecture also makes it certain that the solution can operate in a multi-organization environment. This can be accomplished by simply setting attributes that specify which organization a particular item belongs to. At the same time, it allows multiple users from the same organization to work within the system concurrently.

Another notable approach is that, given the common attributes shared by the different types of operations, an abstract class has been used to encapsulate these shared attributes across the various sub-types of operations. The implementations of this abstract class represent operations in stocks, futures, and options, each with its own specific attributes.

As users require that operations be linked together through chains, a straightforward implementation was chosen: these operations are linked to the operation chain class, which essentially groups operations that can be locked when users request it.

Another important aspect of the solution is the high level of customization available for most classes. Users can customize each market, trading account, broker, or type of commission to their specific needs.

While some of the tables in the database such as market data could be automatically populated by default from external sources, retrieving this data on a large scale can be difficult and costly, and finding a provider that offers all the necessary data could be a challenge. Consequently, the tables are not initially filled with data and the users define the markets, types of commissions, etc. that they need to use in the settings menu.

This solution addresses the diverse needs of users by allowing operations to be linked to markets that trade in different currencies, with funds deposited in trading accounts that may use yet another currency, and commissions that could be in any of these or a different currency altogether.

By using this flexible approach, users will have complete control over the operating parameters depending on the market situation, which guarantees that the system can manage a wide range of circumstances, without the need of adding code. Simpler scenarios might have characteristics that are tightly aligned, which would not require a high level of customization. Nevertheless, the system is still designed to accommodate more complex situations where the scenarios get more complex. The primary goal of this solution is to empower users to define and manage the operability according to their organization’s specific requirements. Such an approach is better as the userbase grows and different customization levels start to appear.

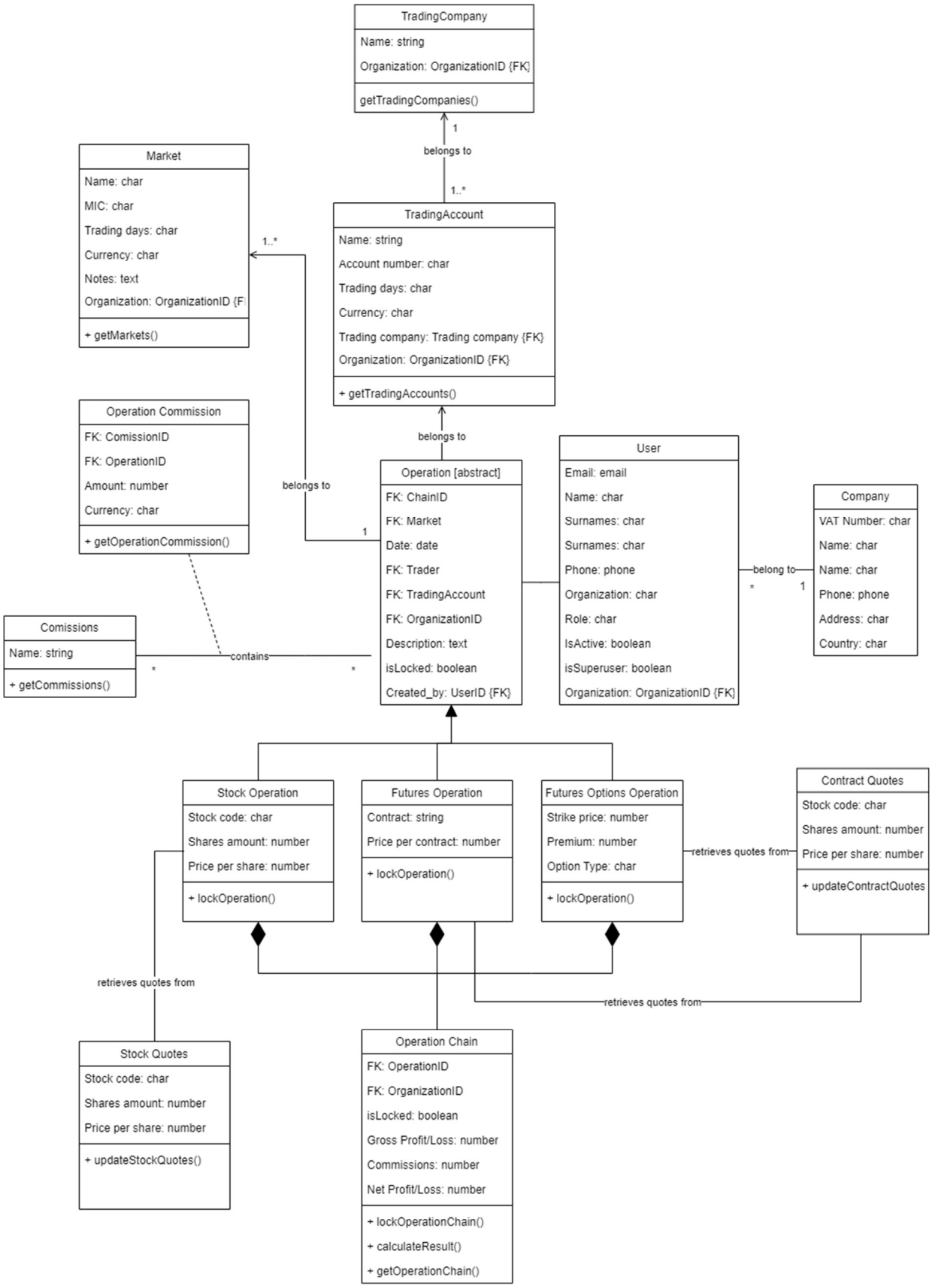

Figure 3: Project UML class diagram

7.2. System Interaction Modeling

Once the workflow has been understood from the user point of view, and the data structure has been defined by the development team, a UML sequence diagram should be included in the project documentation to model interactions between the users and the system. This type of diagram shows a time-ordered view of the system's behavior and the user, reflecting what will be the definitive interaction between the user and the system. Like in this particular case, it should also include the interaction between the system and other external services or applications which are connected to provide data or services. For this project, an external provider is needed to retrieve data from the financial markets, and therefore, that external interaction needs to be represented in the sequence diagram.

The main process flow was outlined in a flowchart in the previous chapter, and the system's static data structure and the relationships between all the data classes are represented in the UML class diagram in Figure 3 in the previous chapter. On the other hand, this chapter's goal is to define the interactions between the various actors and system components now that the requirements and data structures have been well understood through the use of both the flowchart and UML class diagram.

This knowledge is expanded upon by the sequence diagram, which shows how different components interact with one another throughout time. In comparison to what has been defined thus far, it provides a more comprehensive representation of the workflow by capturing the precise sequence of messages that are passed between the actors, objects, or other external systems that interact with the application.

The sequence diagram that can be found in Figure 4 makes certain that the solution complies with both the functional requirements and the design plan by combining the insights from the flowchart and UML class diagram to provide a more detailed understanding of system interactions.

7.2.1. UML Sequence Diagram

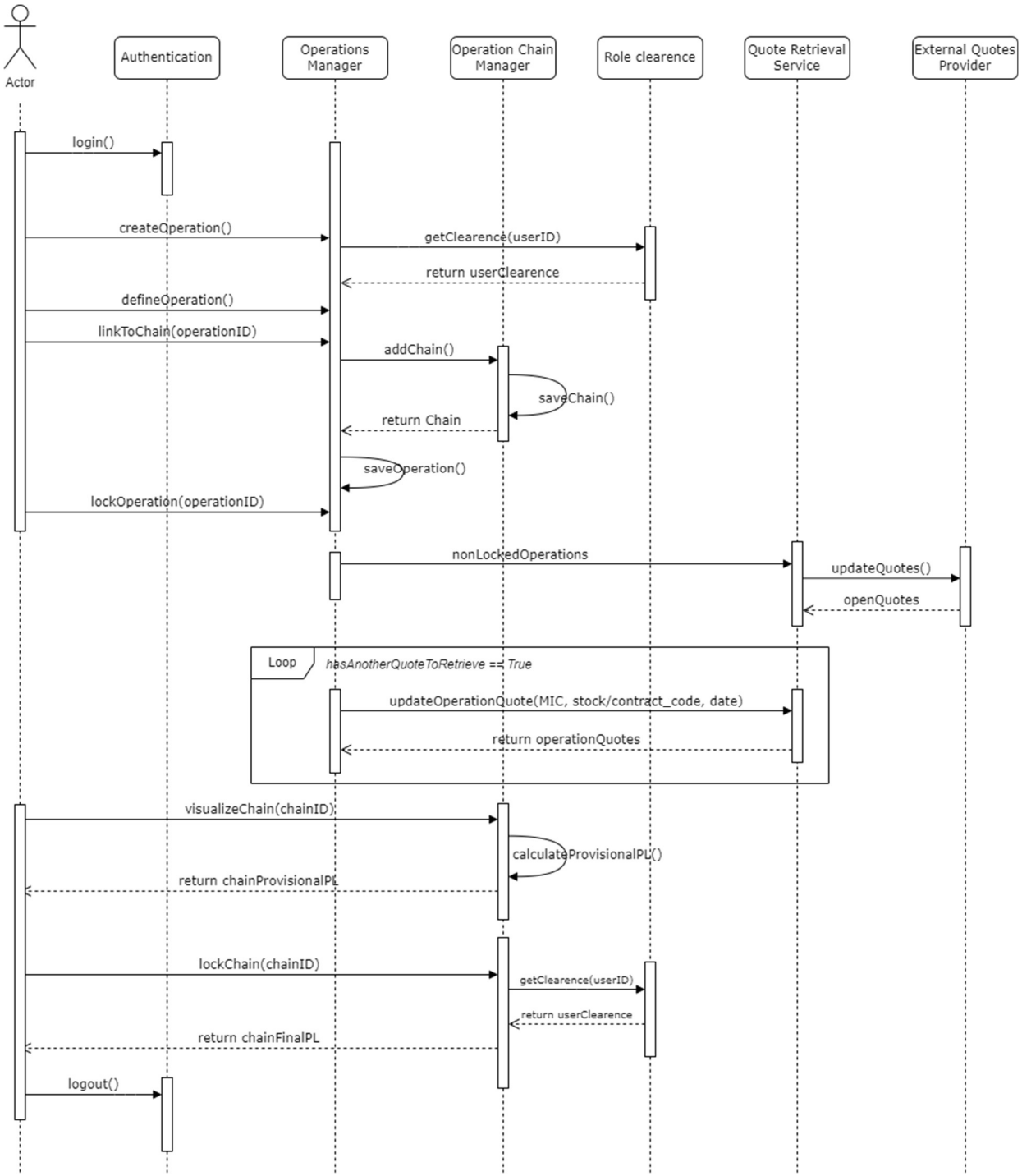

Figure 4: UML Sequence diagram

The most important parts of the diagram are the authentication process needed for the user to access the management functionality of operations and chains of operations, the sequence when the system recognizes all non-locked operations from the different organizations and begins the process to retrieve the quotes from the external provider or providers. In the diagram it is shown that for those positions that remain open, the system will automatically and update the quotes daily belonging to the associated stock or contract in order to reevaluate the provisional profit/loss calculations.

As per user request, this process is triggered automatically daily, meaning that the user will always be access the provisional results calculated by using the quotes from the previous day

Finally, once the user makes the necessary inputs and closes the operations, a definitive result can be calculated and remains static for the future, and the quotes for that position become irrelevant, as they cannot alter the result in any way.

7.3. User Interfaces

For a project that records financial operations to be both efficient and user-friendly, user interface (UI) documentation is very important. In order to help users to locate the different operations, the chains of operations, together with their associated quotes, results, without having to memorize data or go through multiple screens, the user interface for this kind of project needs to be intuitively developed.

Easy access to key elements should be possible with a well-designed user interface. Moreover, a minimalistic design also minimizes distractions and avoids excessive amounts of information that could end up doing everything but helping the user.

The interface can concentrate users' attention on what really important by taking this approach, eliminating unnecessary and decorative features that could worsen the user experience. A well-designed UI should facilitate simple information retrieval and facilitate users to find quickly and easily what they need.

In conclusion, the UI should ensure quick transitions between different parts of the application and should help in speeding up tasks. In the end, that would mean more productivity for the potential customer, and that would eventually make the product more attractive.

7.3.1. Introduction to the Design Approach

The design of the financial operations manager is guided by the need to create an intuitive, efficient, responsive, and reactive solution that simplifies users' tasks to manage financial data, as is the management of stocks, futures and options operations.

Responsiveness and reactivity can only be accomplished if the development of the project is done with a technology that allows for such objectives to be fulfilled, therefore, it is not only a design decision. When it comes to other decisions such as the application layout, menu's structure, accessibility, etc. must be assessed prior to the coding phase to ensure that no reworks need to be done later due to a wrongful implementation or a design that is not optimal when it comes to user experience, which slows down normal operation.

The approach is for the design to provide users with a flawless experience that simplifies the process of managing, tracking, and analyzing these different types of financial operations, with

the goal of registering these as quickly as possible and getting the desired financial results from the chains of operations.

In conclusion, this chapter outlines design strategy, providing a detailed analysis on the factors that influenced such decisions during development. It also discusses the many challenges that were encountered during the design process, to create a user-friendly interface. As every strategy, it comes with advantages and disadvantages which are also discussed in this chapter.

7.3.2. Application Main Design Structure

The proposed application design architecture prioritizes minimalism by incorporating a top-level navigation bar. The top navigation bar follows a well-established design pattern, making it familiar and intuitive to users, because many other common applications use this type of element too.

This design element serves as a centralized hub, providing users with quick access to all of the platform's primary sections. This streamlined interface minimizes the effort required to navigate the application and locate the different menus.

Despite the application's extensive functionality and calculations, the user experience is straightforward since the system handles the majority of the complicated operations without overloading users with complicated forms or irrelevant information that may be difficult to understand.

The application's functionality is centered around several key areas. The home page serves as the initial landing page, providing an overview of different relevant data and provides users access to the source of that data, if they find it necessary to navigate towards it.

Following the home page, there is another dedicated page for controlling all system operations, which includes facilities for creating, modifying, removing, and tracking individual operation statuses.

A specialized page facilitates the visualization and management of operation chains. Operation chains represent sequences of interconnected operations, enabling users to automate complex workflows and calculation that originate from this connection of different operations. The user is able to see the composition of an operation chain together with all of the calculations the system has done for them. The most important results are shown here, which are the results that will have an impact in the decision-making process, such as the provisional profit/loss and the definitive profit/loss that can be used already as a result.

An analytics section can also be used to show the user the different reports the system generates, charts, and any other relevant summarized information that can be useful should be incorporated into the analytics page.

Finally, a configuration page allows users to customize various aspects of the application's behavior, including adjusting preferences such as managing user accounts, and defining organization's markets, traders, trading accounts, etc. These are the attributes that are shared across the organization's users, but that are exclusive to that organization and completely secret to other organizations that might be using the application. Each organization should be able have its own setting.

This layout can be summarized in the following chart:

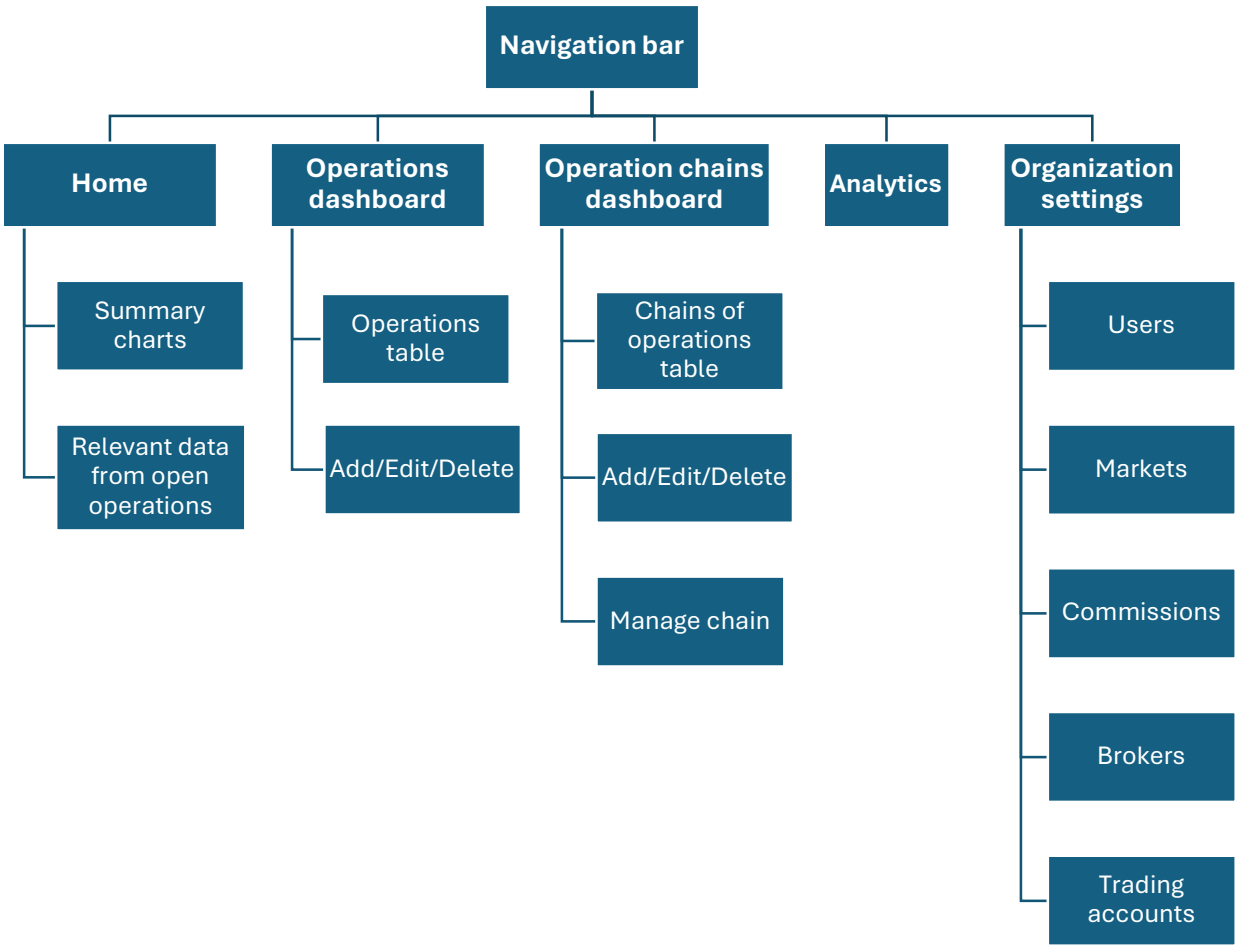

Figure 5: Proposed application design structure

7.3.2.1. Design Components: Navigation Bar

From the user experience side, it can be noted that consistent placement of the navigation bar ensures that users always know where to find the primary controls, and since the navigation bar is always visible, users can switch contexts quickly, which is essential to maximize productivity in this kind of tasks where users will be moving quickly from one menu to another constantly.

7.3.2.1. Design Components: Home Menu

The Home section is designed to provide users with an immediate overview of their portfolio's performance through summary charts and relevant data from the different open operations.

The intention is for this section to act as a dashboard, providing quick access to the most critical information without overwhelming the user with unnecessary details. Summary charts visually represent key metrics, while the relevant data section offers actionable insights about open operations, allowing users to prioritize their attention. Therefore, by summarizing key information and displaying it prominently, the home section allows users to quickly assess the state of their portfolio and identify any immediate actions they need to take. In addition, the use of charts also helps comprehend complicated or large volumes of data.

7.3.2.2. Design Components: Operations Dashboard

The operations dashboard features a table where users can view all financial operations. It includes functionality for users to add, edit, or delete operations. The table format is ideal for presenting large datasets in a structured, easily navigable manner. This design choice was made to ensure users can efficiently manage their operations with minimal effort.

To handle the different types of operations, the table header includes buttons that allow users to switch between operation types. Additionally, there are buttons for filtering, downloading data, and accessing CRUD (create, read, update, delete) functionalities.

By centralizing these tools in one location, users can manage their operations without needing to navigate to different parts of the platform, reducing the time and effort required to perform key tasks. The table’s layout also allows for easy sorting, filtering, and searching, helping users find specific operations quickly.

7.3.2.3. Design Components: Operation Chains Dashboard

The operation chains dashboard is a vital component of the platform, designed to help users manage and track sequences of the related financial operations which get grouped into chains. The dashboard features a table that provides a clear overview of each chain's key properties, such as chain name, status, and Profit/Loss (P/L) calculations. This table format allows users to quickly assess chain performance and main characteristics while being able to filter through all the entries on the table or to download the information shown on the table for them to import that data into another software to process it.

For deeper information on an operation chain, users can easily click on a specific chain to access detailed views of the operations linked to that chain. This approach keeps the main dashboard streamlined while offering detailed information on demand. This simple designs also avoids oversaturating the users with excessive information on the main table.

A key functionality of the dashboard is the ability to finalize chains. Once all related operations within a chain are complete, users can lock the chain, triggering the system to calculate and

record the definitive P/L. This finalization process is simple and integrated directly into the dashboard, ensuring quick and accurate closure of chains.

Overall, the design emphasizes functionality and accessibility, with features like filters for easy navigation, quick access to chain details and a straightforward process for finalizing operations via a simple button.

7.3.2.1. Design Components: Organization's Parameters Panel

To be able to comply with the design shown in Figure 5, a specialized page is dedicated to controlling the Market, commissions, trading companies, trading accounts and user settings of the organization. Each of these parametrizable settings needs to be able to be visualized from one single page, with the ability to add and delete existing parameters.

7.4. Third-Party Integrations: Data Source Connections and Export Capabilities

When it comes to the implementation of a mechanism that retrieves the desired quotes from the different stocks and contracts. It has been decided to implement a mockup to prove the system's capabilities when it comes to retrieving and updating quotes with online data, given the fact that implementing a real connection with an external provider is hard due to pricing issues.

The mentioned mockup consists of an implementation that retrieves values from a made-up response that would be provided to our system from the third party when called should be done. Given that response, the system should process it and update all the values of the open positions according to the response provided by the data provider.

When it comes to the export of our own data, the data that the system has generated via user inputs, it has been decided that the implementation should allow users to export the operation chains table and the operations table. This approach is used by many financial institutions such are banks, with their statements. They commonly allow users to download the data the user is visualizing so that they can process it on other software. In this case, the users should be able to export the data from these two tables in a CSV[3] (comma-separated values) files. That format is read by most spreadsheet processing software and is widely used and easy to read.

7.5. Key Challenges Encountered

[3] Comma-separated values (CSV) is a text file format in which values are separated by commas and records by newlines. A CSV file stores tabular data (numbers and text) in plain text, with each line representing one data record.

A decision had to be made to whether to implement operation chains and operations in the same table and simply use dropdowns to show all the operations that depend on a particular chain, or to use two separated dashboards independent from each other.

In the end, from the user perspective it is also important to identify and deal with operations alone, download them, filter, among other functionalities, it has been decided that the best implementation is to separate them into two different sections as shown in Figure 5.

It is important, from the user's point of view to able to interact with the application and manage operations on their own, without the confusion that could result from combining them with operation chains. When working with the data, having the data separated into two different dashboards can make filtering, exporting much simpler.

8. Development Phase

In this chapter, the main decisions and results from the development phase are discussed and presented. The idea is that there are multiple viable ways of building a solution that complies with all of the requirements mentioned in previous chapters of this document, but this intends to explain the pros and cons of each decision and what have been the results obtained. As a retrospective, non-optimal decisions or errors that may have been committed during the development of the application are also addressed in this chapter as items to improve in future iterations of this project or in other projects as well.

8.1. Introduction to the Development Process

A single developer is in charge to carry out the development phase of this project, as all the other areas this project has covered. It has been decided that the development will be done using a web application created with Django as the backend framework and Vue.js as the frontend framework. The idea is to combine the strengths of both frameworks to create a powerful and scalable solution that does not take too much time or resources to develop.

In the following chapters the reasons behind the decision of employing these two frameworks are presented. The common characteristic of both frameworks is that they are open source, and they completely free to use, which is important given this thesis does not count with a budget to purchase any licenses.

8.1.1. Choosing Django Framework: Main Strengths and Benefits

For this web application, Django is the framework chosen that handles the backend logic, data management, and API creation while Vue.js provides a reactive and interactive user interface.

Even though this project cannot be considered large by any means, and only a portion of what could be developed for the users is within the scope of this project, Django has been considered a very good option due to the fact that is what can be called as a "batteries included" framework. That means that there are some key features included in Django itself that facilitate development of the application very much.

Some of these helpful built-in features are:

- **Administrator Interface:** Django provides a customizable administrator interface that allows the developer to interact with the data of the application very quickly, without the need of coding much. For a solo-dev project, this is an important aspect to consider.
- **Object-Relational Mapping:** Django provides an ORM which allows developers to interact with the database using Python, facilitating the management and implementation of the database.
- **Authentication Module and Sessions:** Django provides with the necessary authentication mechanisms facilitating the user the management of user accounts, groups, and permissions. It also provides full support for session management for handling cookies and control user access.
- **Uniform Resource Locator (URL) Handling:** Django has a simple URL routing system that allows mapping different views (the part of the framework that handles server requests and responses) with the URLs.
- **Security Tools:** Apart from the authentication and session management, Django helps in implementing protection against SQL injection, cross-site request forgery (CSRF) and clickjacking protection.

There are many other features and tools that could be of use in an application such as the one built on this project, like, for example, internationalization features, development tools for testing and debugging. This shows how much the framework actually does for the developer in saving time behind the scenes.

Considering this thesis in total takes 15 ECTS credits, equivalent to 375 hours of time to work on the project (note that this project includes documenting requirements, defining workflows, use-cases, documenting the technical implementation and presenting the results), any help provided by the framework that saves development time is very welcome, and other frameworks, even though they have their own advantages, do not provide this sort of tools that make it so easy to set up the basics of the application.

Besides all of these built-in features, Django has been used in notoriously big projects and has a reputation for being scalable and performant, with a very active community and support, that provides very extensive documentation.

8.1.2. Choosing Vue.js Framework: Main Strengths and Benefits

Vue.js is a JavaScript Framework that has been chosen to develop the frontend of this application due to different factors. One of the most important from a technical standpoint is that because of the strong reactivity system on which Vue.js is based, the user interface will always update automatically whenever the underlying data changes. This feature allows the application to be very dynamic and provide the users with a fast and dynamic interface, ensuring that data remains updated and minimizing user inputs.

When it comes to the composition, Vue.js has a component-based structure that allows many parts of the code to be reused and simplifies maintenance. As the application scales, this is a great advantage. Moreover, these components can be developed in what are called: single-file components (SFC), which are .vue files. A SFC file allows the developer to encapsulate templates, logic and styling in a single file.

Finally, an important part for a novice developer, Vue.js counts with extensive documentation that facilitates the learning process, given that it is a mature framework that has been developed for many years at the time of this project development.

8.2. GitHub: Task Management and Code Storage with Version Control

The code belonging to the project has been stored in a GitHub repository to take advantage of Git's version control system and GitHub's project management environment. Plus, it stores the code safely on the Internet and erases any possibilities of losing parts of the code, while being always accessible from any machine.

Git is a distributed version control system that tracks changes to code over time and allows the developers to view all the changes made to the code, while allowing the developers to revert to previous versions if they need to do so. Even for a solo-developer project like this, the tool is still plenty useful. GitHub essentially provides a platform for hosting the Git repositories online, with some extra features such as issue tracking, pull requests and code reviews, among others.

For this project, using GitHub projects was very useful to write down and keep track of active project tasks and issues. It allows the developer to create, assign and link issues to pull requests. This process allows the developer to streamline project management via dashboard with all ongoing tasks, the progress made on them and related information.

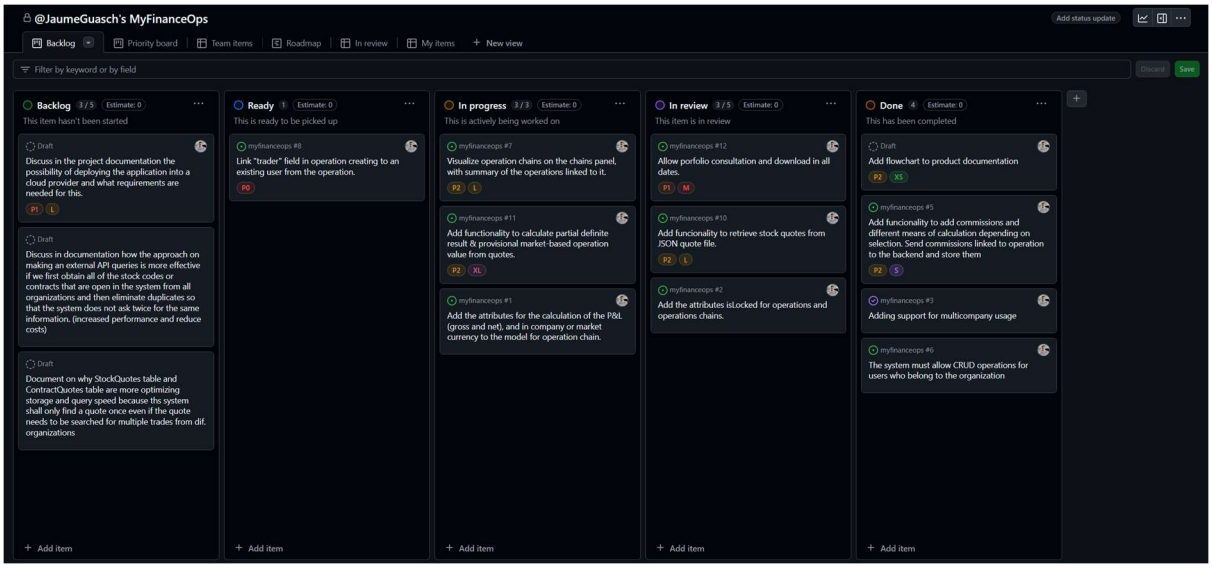

Figure 6: GitHub Projects' backlog

The GitHub Projects backlog has been useful to define many of the tasks and select a priority, for them, an estimated workload, link code to them and set deadlines for each task. It also allows the developer to see a thread with all the different changes an issue has undergone.

For many non-code related tasks in this project, Microsoft Planner has also been effective. Its simpler UI and fewer features make it less suited for code management, but it proved useful for organizing and tracking project-related tasks and activities.

When it comes to storing the project in a repository, for this project a single repository has been created that includes both the frontend and the backend. The reason for not using separate repositories for each component of the application is that it simplifies the development process, considering there is only one full stack developer. If there were a development team on the backend and a different team on the frontend, the more conventional approach would be to separate them into two different repositories. In conclusion, the key has been to try to save as much time as possible, minimizing the overhead of managing multiple repositories at the same time and having a unified view of the project.

8.3. Front-end: Main Points of Interest and Challenges

This section addresses the main points of interest encountered during frontend development, including key issues, the main challenges and the results obtained. It outlines the benefits of using Vue.js as a frontend framework, explaining how it helped resolve development problems and its overall impact on the frontend of the application.

8.3.1. Dependencies Used

The following are the most relevant dependencies used in the frontend together with Vue.js:

- **Vite:** A build tool that improves development speed that also includes a built-in development server with fast hot module replacement (HMR) and optimized build processes. The development server supports automatic reloading of the code, meaning that iterating on changes is quicker and simple. It implements lazy loading of modules, meaning that code is only loaded when it is needed, resulting in improved performance.

- **Vue Router:** It is the official routing library for Vue.js, allowing for the management of navigation between different components and pages.

- **Pinia:** A state management library for Vue.js that replaced Vuex. It makes state management simple and provides Typescript support (which has been used in this project in exchange for JavaScript, due to its offer of static types). It has been used especially for managing authentication and session management.

- **Tailwind CSS:** It is a utility-first CSS framework that allows for styling of components with design consistency and responsiveness. Tailwind CSS provides many utility classes that help build custom designs without the need for custom CSS code.

8.3.2. Front-end Results

The landing page of the application looks like Figure 7. As a curiosity, a made-up logo has been created using an Artificial Intelligence tool that helps in that type of task and makes the application a little more appealing for showing it in this documentation, even though obviously, it does not add anything to functionality.

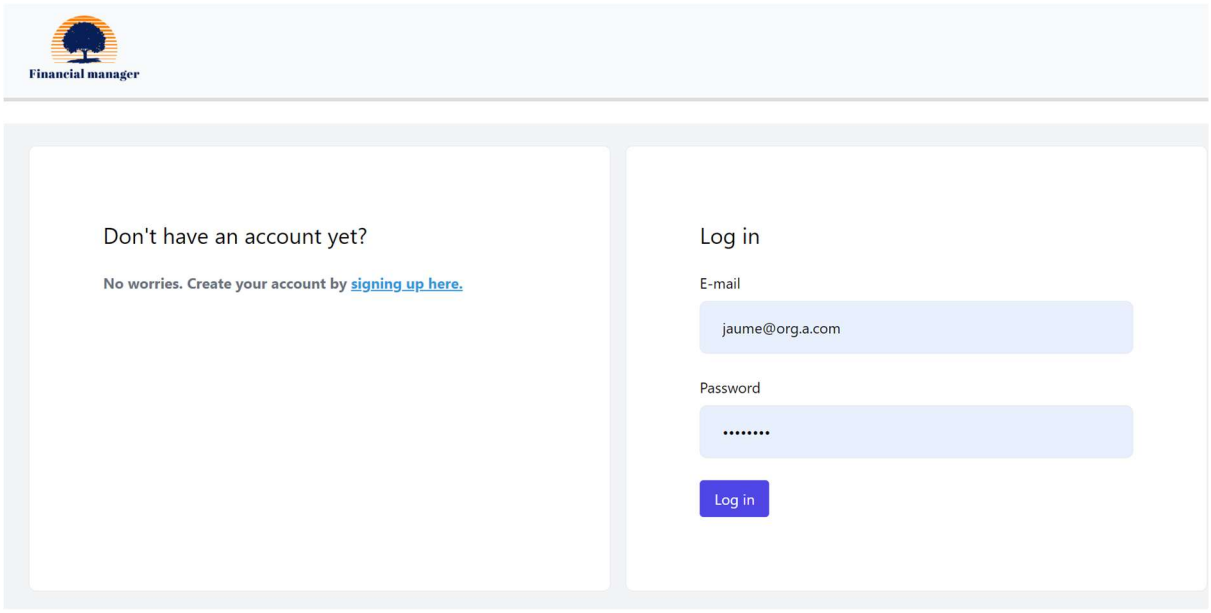

Figure 7: Application landing page

The signup page has been developed for testing purposes, even though in a final version of the application it would be removed, and signup would only be allowed by an authorized member of the organization.

To comply with the design of having a navigation bar on the top of the page that is persistent throughout the navigation around the application, and that gives easy access to the different main sections of the application.

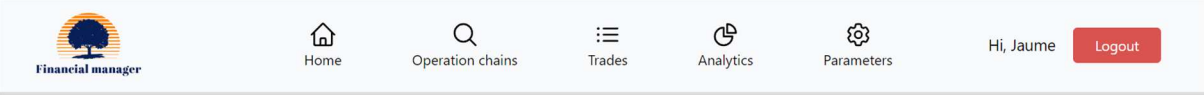

Figure 8: Navigation bar

Most elements in the navigation bar remain hidden while the user has not been authenticated into the application, only showing the logo and the separator.

Once authenticated, one of the first tasks users will need to do is configure or set the parameters that they need to be able to register trades in the application. Without these parameters, the system would not be able to retrieve data from the markets or perform its calculations.

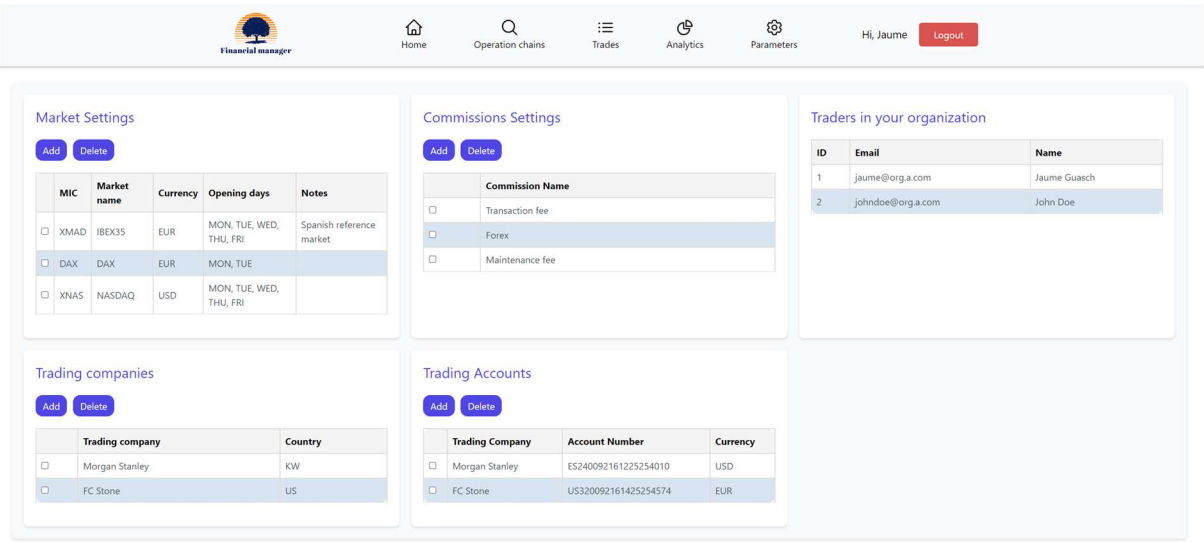

Figure 9: Parameters page

According to the design schema, users control all the parameters that are unique to their organization from this single page. Design is consistent with the requirements, tables have been used to display information and two buttons are used for each table, add and delete, used to register new entries in each table or to delete them, if possible.

As an example, all of the forms used to create new entries in each table look like the one shown in Figure 10. They employ a pop-up system where all the necessary fields are required and validated. The use of dropdowns to input currencies, countries, or trading companies, which are necessary to register a trading account, minimizes potential errors during user input and

sends the backend standardized codes in the HTTP request (such as ISO currency codes or country codes, for instance) so that data is consistent and harmonized.

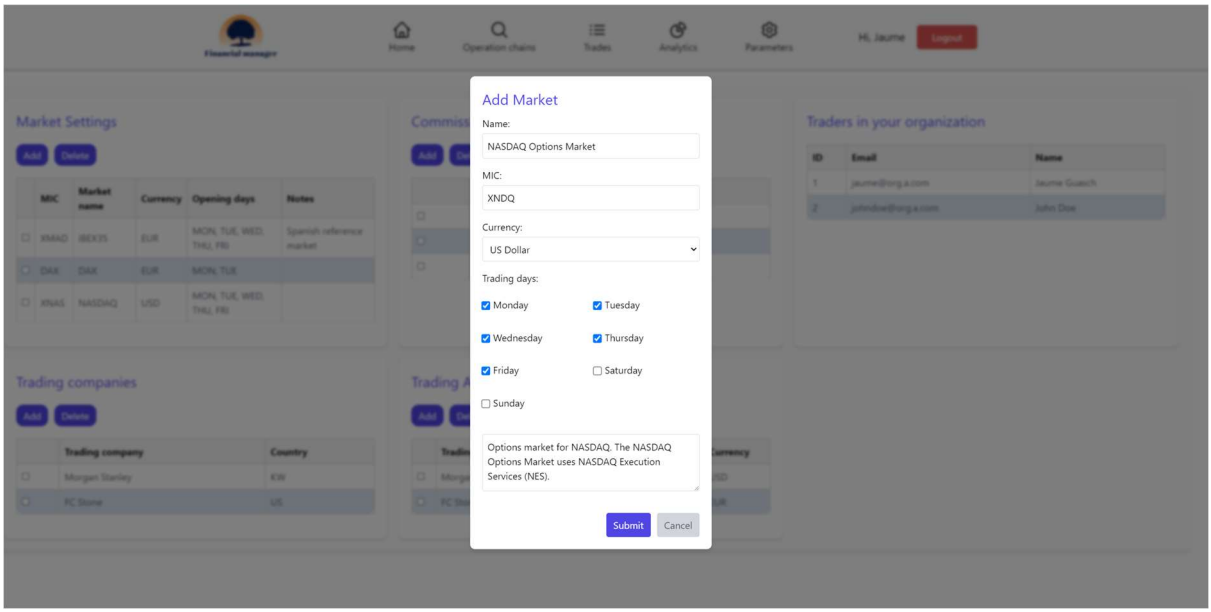

Figure 10: Form example for parameter data submission

To display all of the operations, an operation dashboard has been created with three main elements.

Firstly, the header contains buttons that allow users to switch between filtering stocks, futures, and options. Since these financial instruments have different attributes and are not linked to each other, they are displayed separately according to user requests.

The header also includes three buttons for downloading only the visualized data, providing users with options to export their data in various format, a filter, that allows users to perform searches and hide columns they might find unnecessary, customizing the view to meet their needs and the ability to create a new operation from scratch.

The last important element to distinguish is the table itself, which updates according to the selected type of operation and the filters enabled by the user. Figures 11 and 12 show the end result, an approach that is minimalist, simple, customizable, and, thanks to the reactivity that Vue.js offers, quick for searching and filtering.

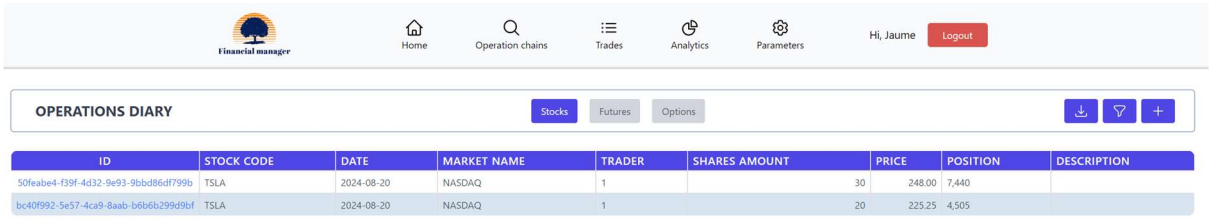

Figure 11: Operations table with the Stocks options selected, and no filters enabled

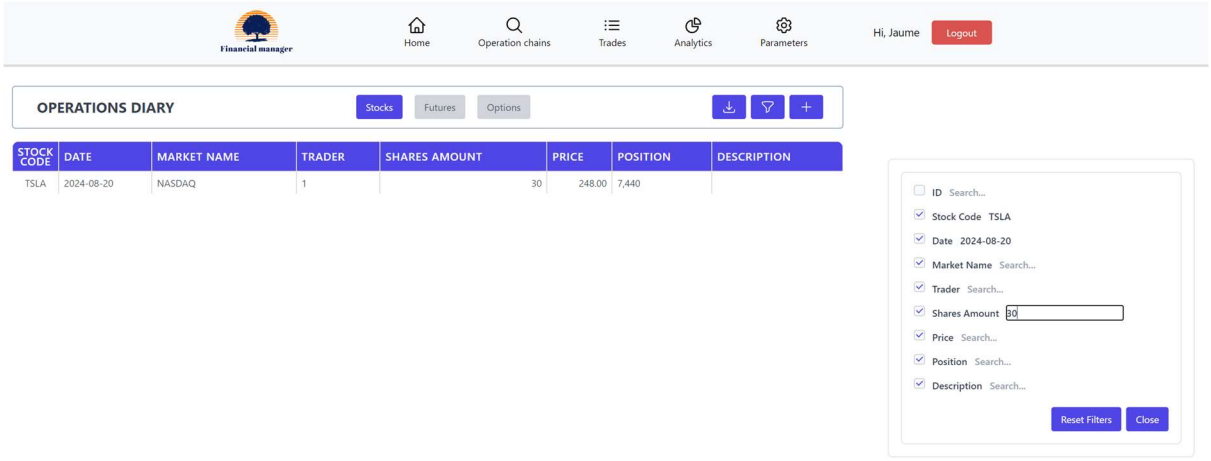

Figure 12: Operations table with filters enabled

Notice how in Figure 11, columns can be shown or hidden, and how immediate inputs in the search bar on in checkboxes apply their effects immediately, without the user having to refresh through a button click as it happens with many other applications. This is the advantage that reactivity gives, it makes the user much more productive by saving manual inputs.

In a similar approach, the dashboard that displays the operation chains and its relevant information has been designed and implemented in a very similar form, as shown in Figure 13. With similar functionality and almost the same layout in order to unify the design and make it user-friendly, as it stays in harmony with the rest of the application.

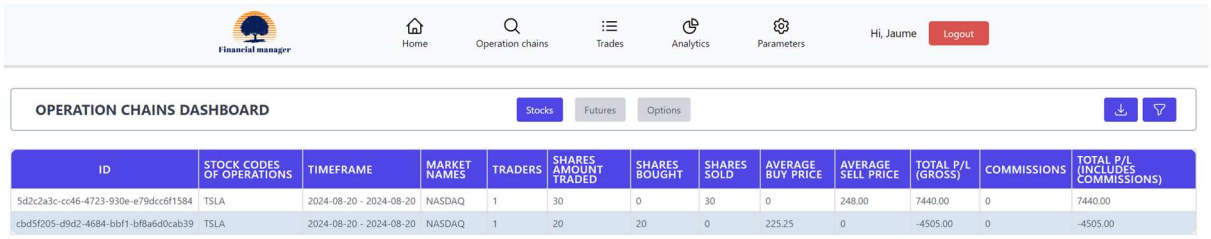

Figure 13: Operation chains table

8.4. Back-end: Main Points of Interest and Challenges

The primary topics of interest during backend development are covered in this section, along with important problems and final results. It describes the advantages of making use of Django as a backend framework, including how many of its built-in features helped overcome difficulties in development and what effect it has had on the backend as a whole.

8.4.1. Back-end architecture

Django provides developers with apps, which are reusable and can be used in multiple projects. It is what is defined as a “pluggable architecture”, meaning that apps can be reused into different projects. Each app has its own models, views, templates, etc.

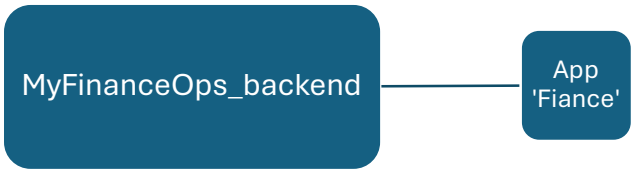

Figure 14: MyFinanceOps_backend is the project, and Finance is an app that is being used in this project.

This allows for the app ‘finance’ to be reused in another project if necessary. This degree of modularity allows developers to keep their codebase organized and maintainable.

8.4.2. Django Rest Framework

In order to communicate with the frontend, the Django Rest Framework is used within the backend.

The Django Rest Framework is used for building RESTful APIs, which is necessary to facilitate communication between the frontend and backend in this application. It does so by expanding Django's functionality. DRF makes it easier to provide access to Django models and business logic through API endpoints, allowing clients to communicate to the application over HTTP. That could include web browsers and mobile apps and other servers.

Moreover, it includes support for serialization, which transforms the complicated data types from Django models into JSON that can be easily transferred to the frontend server. This is one of DRF's primary features. DRF also offers throttle, permissions, and authentication management tools, which simplify the process of protecting the APIs and managing access.

8.4.3. Django User Authentication and Sessions Management

A mechanism for user authentication is included with Django with its 'django.contrib.auth' module. It manages cookie-based user sessions, user accounts, groups, and permissions. The Django documentation covers how to implement all of these features into a project such as the one developed in this thesis.

For this project, the user model that comes with Django was not considered to be good enough and not have the proper attributes needed for authentication. The intention is that the

organization email is used for the authentication together with a password. Therefore, a custom user model was defined in the models (using the AbstractUser abstract base class), with the necessary attributes.

After the user model is defined, use of 'rest_framework.authentication.TokenAuthentication' module has been made. These modules together with the Token Authentication allow the Django Rest Framework to check credentials against the database upon a log in attempt. If correct, the system will respond with an authentication success message to the frontend and will assign a session id to that user session. The session id is sent in every HTTP request from the frontend which ensures that the user making the request in the first place has been properly authenticated and should have access to the API endpoints.

User sessions are managed by the `django.contrib.sessions` module of Django. These user sessions are necessary for preserving state between requests in the web application. Each user is assigned a unique session ID by this module, which also stores the associated session data on the server and stores the session ID as a cookie in the user's browser. With the use of this mechanism, the application can monitor user activity and store information about them, including their preferences and authentication status.

8.4.4. Django Admin Panel

One of the built-in features that comes with Django is the Django Admin Panel contained in the 'django.contrib.admin' module, as mentioned before. This consists of a control site where, especially during the early phases of development, developers can test models, relationships and manage other database content. It does so by automatically providing a visual interface that allows the developer to perform CRUD operations on the existing models.

Not only it can be used for development, but organizations could also rely on the Django Admin Panel as an internal administration tool. It has a safe and regulated environment for non-technical users, where they can interact with the application’s data. Authentication is required to access the administrator site, and only authorized users can perform certain operations. This avoids the need for a dedicated administrator panel in the early stages of an application, especially if the intention is just to monitor the data and perform CRUD operations.

The Django Admin Panel is also customizable, offering a default interface out of the box, but can also be adapted to fit specific needs if the project demands so. Admin views, forms and templates can be adapted to deal with the business processes of to provide extra functionality that the default interface does not support.

For this project, the admin panel has been an extraordinary, especially when the frontend is still in development and the admin panel remains as the only tool to test out the basic CRUD operations of the models and their relationships.

8.4.5. API endpoints and Server-side Logic

Within Django, as mentioned earlier in the introduction, a URL can be linked to a view or a function. In this project, the URLs within the `finances` application serve as the API endpoints. This is achieved by including the application's URLs in the main project URL file with the line `path('api/', include('finances.urls'))`.

The intention is to create endpoints that are not only used or called by our own front-end, but also to have endpoints that would be capable of integrating our application with external systems. For instance, a possible use-case for this scenario is that the potential clients of this application might want to connect their ERPs or accounting software with our system. Our system would need to provide an API that is accessible to them, and they would use this API to access and retrieve relevant information to process it in their own system.

A clear example would be the automatization of the registration of profits and losses suffered in the financial markets during their normal operativity, account for commissions spending or generating accounting entries in a fully automated manner. This is the kind of integration that would save a lot of time in manual data inputs and minimize mistakes, as data would only be entered once in this project's application and then be transferred thanks to the use of an API into an external system.

Ideally, in a more ambitious and lengthier project, this could be represented by an API mocking tool such as Swagger, which helps building dynamic mock APIs that the development team can use to replicate API interactions, which are a simulation of the functional, real service.

8.4.5.1. Django Views

The URLs in the application then are linked to a function that is defined in the views of the application. To summarize the functionality of Django's views, they handle the logic of processing requests and returning responses, typically rendering an HTML template or returning data in formats like JSON for APIs. The most common HTTP requests the API will handle include GET, POST, PUT, PATCH and DELETE requests. The views ensure the type of request coincides with the HTTP request the functions are meant to handle, and that the user making that request has permission to actually perform it on the back-end side.

An extensive number of functions has been created to perform CRUD operations with data from the database and return it to the frontend when necessary, making sure restrictions apply. Some of these restrictions include, verifying permissions and authentication.

In this case, there must be special care when dealing with requests to make sure that in the response data from different organizations from the request's is not sent with the response. Data must be carefully filtered and send only that organization's data. This is done thanks to the fact that all the tables that contain sensible information have a specialized attribute that is the organization's unique identifier, as shown in the data structures section of this documentation, which ensures that an entry in the table belongs to a certain organization. Then, when the system processes a request, it can make sure to return, modify or delete matching entries in the tables, and not access or modify any other organization's information.

8.4.5.2. Django Serializers

The Django Rest Framework (DRF) offers utilities called Django Serializers that help with the easy representation of complicated data types, such as Django models, in APIs by converting them into JSON, XML, or other content formats. Additionally, they take care of deserialization, which is the process of converting incoming data back into complex types so that it can be verified and stored in the database.

8.4.6. Database

For the needs of the data structures design and the data modeling solution that has been adopted, a relational database is what has been needed to implement.

For development purposes, SQLite3 has been used, which is the default database in Django. The decision not to change comes after deciding that its lightweight nature, simplicity and easy setup was enough for this project during the development phase. Other databases like PostgreSQL, MariaDB, MySQL and Oracle are supported by Django.

On the other hand, PostgreSQL would be a better option for production deployment, since it provides sophisticated capabilities that are useful for managing bigger volumes of data and more complex operations, such as support for sophisticated queries. Enhancing query performance would be important in a production environment, especially as the data grows. PostgreSQL has support for advanced indexing and full-text search, which improves query performance.

Furthermore, PostgreSQL outperforms SQLite3 in terms of concurrency handling and scalability, which increases its dependability in production settings where numerous users may interact with the database at once. Its broad support for both transactions and data types further guarantees its ability to handle a wider range of data requirements, in case they were needed in future developments.

8.4.7. Market Quotes Retrieval System Design

One of the challenges of this application is the fact that some calculations need to be done with data that is available online through external market data providers. Such information needs to be retrieved on a daily basis and used to make certain calculations that affect the projected profit/loss.

The projected profit/loss varies constantly according to the markets, but for accounting purposes an idea was suggested by users to use the end-of-day quotes as a reference to make all calculations.

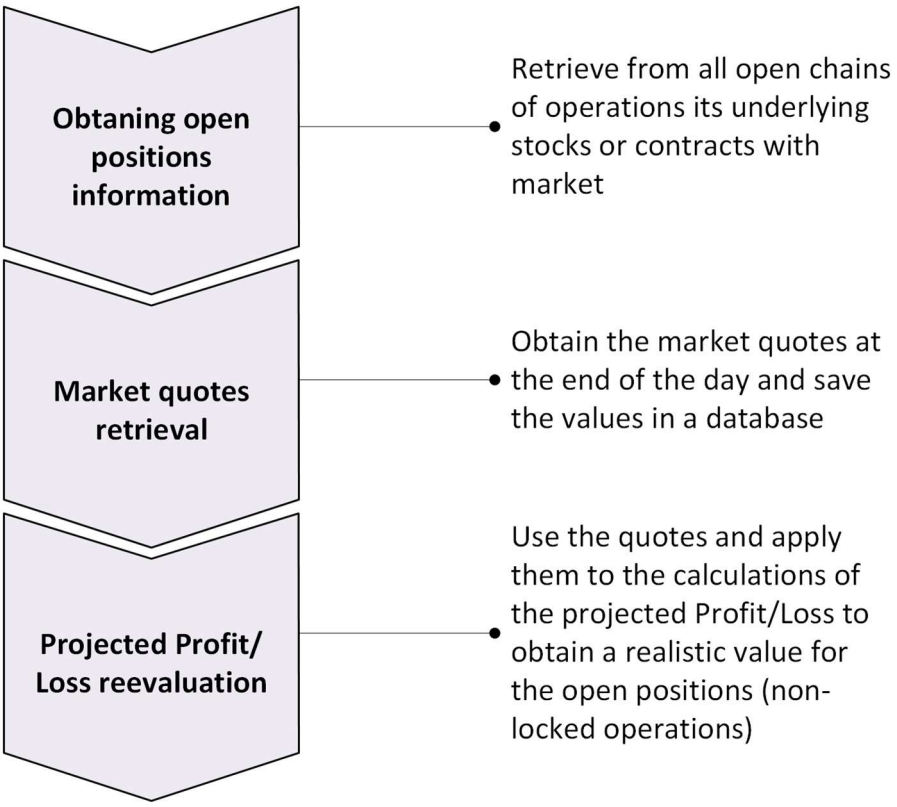

Figure 15: Process of retrieving market quotes to update system information

Operations' calculations need to be performed in different ways according to the types of operations, whether it is a stock, futures or options and the currencies used in the markets and the trading account.

In the following chapters the retrieval mechanism according to the type of operation is explained, together with data samples and an explanation of the mockup case set up to test the mechanism in question.

8.4.7.1. Stock Market Data Retrieval

When a user defines an operation with a stock, and the user has not terminated that position, meaning that:

- **Operation chain status:** Non-locked
- **Operations within the operation chain status:** Non-locked

Such state is monitored through an attribute that exists in both chains of operations and operations themselves (see Figure 3 for more detail).

The quotes for the stock belonging to the open position will change in value over days. The stock quotation will be obtained by a third-party. An example of a quote provided by an external market data provider could be the following:

```
{
   "symbol":                "AAPL",
   "companyName": "Apple Inc.",
   "exchange":              "XNAS",
   "date":              "2024-08-21",
   "open":                  226.4287,
   "high":                  258.7714,
   "low":                   221.3305,
   "close":                 251.2959,
   "previousClose":         226.3471,
   "change":                 24.9488,
   "percentChange":          11.0223,
   "volume":                99636770
}
```


Figure 16: JSON content showing an example of a quote provided by a third party for a certain day, stock, and market.

To generate a mockup scenario, massive quotes have been generated for multiple stocks across different markets and on multiple dates following the format provided in Figure 16.

Most external providers that have been researched require a stock code and a market to be provided in the request to be able to locate the correct quote. The result is that they return a single quote, for the current date and a particular market, based on request parameters.

Other functionalities exist that could be used to retrieve stock historical data, but the key point of this approach is to always have the latest data available from the provider, and count with the last end-of-day quote that exists.

Once the JSON has been obtained, the data must be processed and stored in order for the system to be able to retrieve it when necessary. These quotes are stored in tables *StockQuotes* and *ContractQuotes* according to the proposed architecture design.

The code that would take care of this part of the functionality would be as simple as Figures 17 and 18 show:

```
def update_all_stock_quotes():
    print("Updating stock quotes...")
    unlocked_operations =
StockOperation.objects.filter(isLocked=False)

    for operation in unlocked_operations:
        stock_code = operation.stock_code
        market_identifier = operation.market.mic

        # Call the create_stock_quote function
        result = create_new_quote(stock_code,
market_identifier)

        if 'error' in result:
            print(f"Error for {stock_code} in
{market_identifier}: {result['error']}")
        else:
            print(f"Stock quote created successfully for
{stock_code} in {market_identifier}")
```

Figure 17: Process that retrieves all non-locked operations and obtains their stock code and market identifier to call the function described in Figure 18

The process shown on the figure above detects all operations of type stock, which are unlocked, which means that its value must be calculated with a retrieved stock quote from the Internet.

The necessary attributes to be able to find the correct stock code amongst the potential indefinite number of quotes are the stock code, and the market code. For the date, the system always considers as the current date as the one that will be registered in the quotation tables.

The process of reading the JSONs and registering the quotes on the tables can be seen on Figure 18, with some validations in place to avoid duplication when registering the codes, as that would increase the data load and decrease performance. This means that if a stock or a contract exists in multiple operations, its value should be retrieved from the provider only once and should be saved in the tables that store the quotes only once too, otherwise, multiple entries containing the same data would be registered. That, multiplied by a very large number of organizations accessing the same quotes could degrade performance and fill the database with redundant information.

It is remarkable that even though these tasks have been scheduled using the Django schedule package, in a real-world scenario it would be preferable to use a tool such as Celery, or any other distributed task queue system that would allow the system to perform this and other asynchronous tasks in a much more sophisticated way. Given this consists of only a mockup, a simpler approach has been taken in order to show the potential of the market retrieval tools developed, without focusing on the scalability and the collection of work result.

In conclusion, the intention when accessing an external system to retrieve quotes should be to filter all duplicates and make sure only one request is made for each stock or contract and

market for the same date. The same mechanism then applies to the process of registering the quotes in the database as explained in the previous paragraph.

```
def create_new_quote(stock_code, market_identifier):
        directory = os.getenv('STOCK_QUOTES_DIR')
        if not directory:
            raise EnvironmentError("Environment variable
'STOCK_QUOTES_DIR' not set")

        current_date = datetime.now().strftime('%Y-%m-%d')

        filename =
f"{directory}\\{stock_code}_{market_identifier}_{current_date}.json"

        if not os.path.exists(filename):
            return {'error': 'Stock quote for the current date not
found'}

        with open(filename, 'r') as f:
            quote_data = json.load(f)

        # Check if a stock quote with the same date, stock code, and
market already exists
        if StockQuote.objects.filter(symbol=quote_data['symbol'],

date=datetime.strptime(quote_data['date'], '%Y-%m-%d').date(),

exchange=quote_data['exchange']).exists():
            print(
                f"Stock quote for {quote_data['symbol']} on
{quote_data['date']} in {quote_data['exchange']} already exists.")
            return {'message': 'Stock quote already exists'}

        # Create a new StockQuote entry
        stock_quote = StockQuote.objects.create(
            symbol=quote_data['symbol'],
            company_name=quote_data['companyName'],
            exchange=quote_data['exchange'],
            date=datetime.strptime(quote_data['date'], '%Y-%m-
%d').date(),
            open=quote_data['open'],
            high=quote_data['high'],
            low=quote_data['low'],
            close=quote_data['close'],
            previous_close=quote_data['previousClose'],
            change=quote_data['change'],
            percent_change=quote_data['percentChange'],
            volume=quote_data['volume']
        )

        return {'message': 'Stock quote created successfully',
'stock_quote_id': stock_quote.id}
    except Exception as e:
        return {'error': str(e)}
```

Figure 18: The process that obtains the stored JSON the market, stock code and date of which match with the non-locked operation's parameters passed in the function

Once the process of retrieving and storing all the necessary quotes in the database has finished, a process runs through the open positions, and with the date, the stock or contract code and the market identifier it is able to update all the operations with the up-to-date values, which resemble reality and provide users a clear view on the open positions state.

Such process is simple given that the system only needs to use the current date and the desired value from the table (usually the share price or the contract price when dealing with futures or options) and use that value for the calculation of the Profit/Loss according to the functional requirements (see chapter 5.2.6 for the detailed formulas). Nevertheless, at this stage, all information is available to perform the calculations.

8.4.7.2. Futures and Options Market Data Retrieval

The exact same process as the previous chapter is used to retrieve market information from the servers considering that there are some different attributes such as contract code instead of stock code, contract quote instead of share price, and the existence of strike price and premium quotes for the options markets.

This chapter does not go over the stages in detail because the processes and the data used are very similar to those covered in the previous chapter. It has been deemed as unnecessary since the processes employed for the retrieval of futures and options market data follow a similar structure and logic even though the formulas and attributes may vary slightly.

8.5. Currency Exchange Rates Retrieval

A similar but lighter mechanism but as the ones described in pervious chapters is needed when dealing with the case where market currency does not match account currency. Only in such scenario this mechanism is used and intervenes in between the market quote retrieval process as Figure 19.

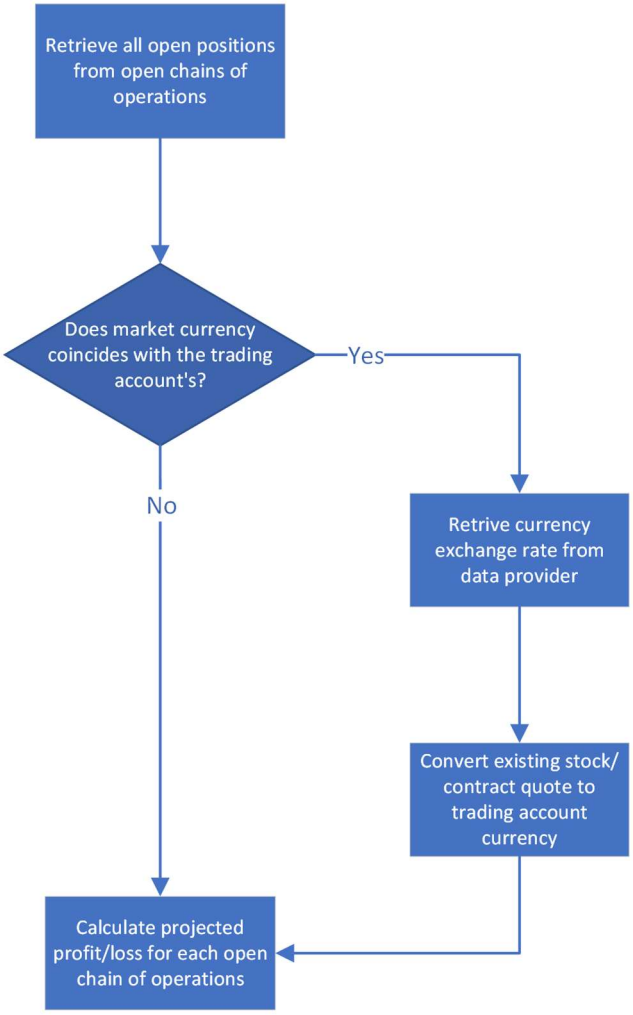

Figure 19: Currency exchange rate retrieval and intervention in profit/loss calculation workflow

The user can provide an exchange rate for the currency at the time of buying or selling but is completely impractical that the user updates that exchange rate on a daily basis while the position remains open.

Therefore, the system must be able to convert the quotes retrieved from the market with exchange rates retrieved from the foreign exchange market, known as Forex[4].

The mechanism is similar to the previous ones defined, but in this case, instead of retrieving the stock code of the open operations, only the market currency and the trading account currency is needed.

A request is made to the data provider with both parameters and an exchange rate is returned.

[4] The foreign exchange market, often known as the currency market, fx, or FX, is a worldwide decentralized over the counter (OTC) market where currencies are traded. The foreign exchange rates for each currency are set by this market. It covers every facet of purchasing, selling, and converting currencies at established or current rates.

The retrieved quote, which is certainly going to be defined in the market currency is then converted to the trading account currency and the projected Profit/Loss is successfully stated in the trading account currency.

9. Future Directions and Expansion Opportunities

As the project development ends, it is important to identify potential areas where upgrades might be needed in the future. This chapter focuses on presenting opportunities for new developments on the project. With these possibilities in mind, a roadmap could be established to define the next stages that wait for this project.

9.1. Cloud Deployment

The deployment of a web application to a cloud provider offers multiple advantages such as scalability, availability, automatic maintenance, major security improvements and backup recovery systems that are far more robust than the ones found in on-premises solutions.

This means that it depends on the needs and the number of concurrent users on the application, it could be worth to invest time and resources to deploy the application on the cloud. Once the deployment is complete, it might actually save time due to all the tasks the cloud provider carries out.

Nevertheless, carrying out a deployment of an application such as this can be complicated and would require an entire subproject where all the requirements need to be well defined in order to carry out the migration.

9.2. Integration with ERPs or Accounting Software

One of the reason this project is useful is that it synthetizes information that is stored on many different platforms belonging to multiple brokers, or trading companies. This information needs to be consolidated before it is ready for being registered into the accounting system of a company.

The more brokers or trading companies an organization uses, the more complicated it is to keep track of the different operations and its associated commissions. One of the most ambitious features that could be developed within this application and the client's ERP or accounting software would be automated accounting entries for commissions spending, revaluation or depreciation according to the value of the open positions and registering the gains and losses sustained in the financial markets.

The system would be able to do so my importing the operation chains, together with all of the operations linked and the associated commissions. By filtering on the dates, the system could

easily process the information. As was noted during the user interviews, one of the most mechanical and exhausting tasks is to register these operations into the accounting system, and even more to keep track of all the commissions associated with all the trades, as there might be dozens or even hundreds of trades being carried out on a daily basis, each of them with their own commission spending. Commission spending might also want to be tracked by the type of commission through a budget plan or via cost centers.

This could be fully automated as the system would recognize the type of operation or the type of commission and create new accounting entries for each type, with a predefined budget account or a cost center. It would also incorporate the date and trading accounts automatically, to make sure that treasury control is well managed.

9.3. Integration with Brokers or Trading Companies

The same process explained in the previous chapter could also be used when creating new operations with their associated codes, markets, amounts, commissions, etc. Manual data inputs would be drastically reduced if the statements from the broker could be imported and processed on a daily basis by the application.

During interviews conducted with users from a different company to where this project was developed, it was suggested by financial staff that they had automated statement imports into a spreadsheet to manually process it there.

This process could be taken a step further, and instead of importing the statements into a spreadsheet, the statements could be read and processed by the application itself to fill up most if not all the necessary information to register new operations, depending on the type of operation and context.

The obvious downside is that there is no global standard for these statements and each broker has a different mechanism used to export these, with different files being used with different formats altogether. This means an individual integration needs to be done with each trading company the organization works with, extending the development time.

10. Methodology: Utilization of Project Management Tools

For the development phase of this project, GitHub was chosen to manage this project by using a single repository, which contained both frontend and backend components. This configuration allowed for version control and effortless integration, which was a priority considering the time limitations of such an ambitious project.

Throughout the development process, tasks were organized and tracked using GitHub Projects and issues. A Kanban-style board has been used to monitor the tasks that surged from

functional and non-functional requirements. The board provided a visual representation of each task's status at each step, including To Do, In Progress, In Review and Done. The ability to link each activity to issues with pull requests makes it easier to track development on each of the issues.

Even though GitHub with its GitHub Projects functionalities serve as a powerful Agile tool, with other tools apart from version control that include code review, issue control, DevOps integration, etc. the truth is that this project management could not be considered Agile as it only made use of some of these tools. Agile project management would be much more powerful and necessary on a larger project with more developers which is meant to be deployed. This is not the case as the implementation is only a small part of the thesis. Agile typically involves iterative development with regular sprints, testing, and user feedback. However, this project did not incorporate these elements. The focus was set on completing the already large volume of work, exceeding the three hundred hours dedicated to design, document and implement functionalities.

User feedback would be a part of the process that could follow next, as part of an iterative development, like Agile methodologies suggest. The product should now be presented to the stakeholders to obtain feedback and implement upgrades in future developments. In this case, the project aimed to deliver a finished product based on the pre-defined requirements with no changes during the development phase.

11. Conclusions

This thesis has been an enriching educational experience that has provided me with deep knowledge and improved my skillset in a variety of project management and web software development. It has helped me improve in my requirements collecting skills, as when I was working as a consultant, I learnt how to ask the right questions, and read between the lines of user responses to define clear and unambiguous requirements.

Throughout the project, I improved my full-stack developer skills while using Vue.js and Django. It must be said that the first one was unfamiliar to me and I decided to choose it as many developers both on the online and in person recommended it to me for its gradual learning curve and intuitive development project, and I must admit they were right. For Django, its extensive built-in feature set proved to be an invaluable asset. These are the types of features that save dozens of hours of development time, which is something that might not seem much important as a student but become more important as a developer starts their career and the need for meeting deadlines becomes real. Overall, Django has proven to be an effective framework and would use it again for more projects to come.

The practical experience of gathering requirements from stakeholders in a real-world environment showed me the complexity of software development. The process was demanding but became more manageable as I became more experienced. A significant advantage was leveraging knowledge from my Business Management degree, and more particularly, from my thesis, which enriched my knowledge in the project topic. This background was particularly important when dealing with user interviews, as I was able to steer conversations towards the most relevant points and those that were the hardest to define, and even to identify mistakes from users' responses. I observed that lacking domain knowledge led to an excessive reliance on a potentially misleading user input, even when using the right interview methodology.

This project not only has improved my skillset in software design and development, but it has also inspired me to explore new areas of interest not covered in this project. Some of these areas of interest include DevOps, cloud application deployment, which seems to be the way to go in current times with most applications and understanding how to build CI/CD pipelines, necessary in bigger projects that continuously evolve. All of this together would give me a solid foundation for building successful web applications.

Finally, I should mention that this thesis project has encouraged me to take bigger challenges in this field. The ease of learning and working with these technologies has inspired me to explore the creation of additional projects that might be of interest to others.

12. Bibliography

Django Project Documentation, Version 5.1, <https://docs.djangoproject.com/en/5.1/,>
Date of consultation: 1/7/2024

Vue.js Documentation, Version 3. < https://vuejs.org/guide/introduction.html>
Date of consultation: 1/7/2024

Django REST framework Documentation, Version 3.15.2. <https://www.django-rest-framework.org/>
Date of consultation:2/7/2024

Pinia Documentation. < https://pinia.vuejs.org/>
Date of consultation: 8/7/2024

Coll Caballero, Jordi. UML 2, from Software Engineering II.
< https://imae.udg.edu/~sellares/EINF-ES2/Present1213/UML2.pdf >
Date of consultation 15/7/2024

Bharath Padmanabhan. Unified modeling language (UML) overview, 2011.
< https://people.eecs.ku.edu/~hossein/810/Readings/UML-diagrams.pdf>
Date of consultation: 16/7/2024

Celery Documentation, Version 5.4. < https://docs.celeryq.dev/en/stable/django/first-steps-with-django.html>
Date of consultation: 14/8/2024