POLITECNICO DI TORINO

Master's Degree course in Computer Engineering

Master's Degree Thesis

Design and development of a system for the analysis of financial options

Supervisor
prof. Alessandro Fiori

Candidate
Klaus Cuko

October 2021

Acknowledgements

I would like to thank all the people who supported me not only in this last months, but also throughout my university years.

Firstly, I would like to thank my thesis advisor Prof. Alessandro Fiori, who gave me the opportunity to work on an interesting project and allowed me to express my strong interest in data analysis and software development. He mentored me with his helpful comments and advice, pushing me to do my best. I would like also to express my gratitude to the to professional trader Marco Rossi who has helped me better understand the financial world with his huge experience.

I thank all my friends, with whom I shared a lot of beautiful moments.

Finally, I must express my very profound gratitude to my family that has always believed in me and encouraged me to never give up and to my girlfriend, Serena, for providing me with unfailing support and continuous encouragement. This accomplishment would not have been possible without them. Thank you.

Alla mia famiglia

Abstract

The thesis focuses on studying and analyzing the financial market with particular interest in derivative financial instruments such as futures and options. After that it deals with the design and development of a web trading system able to meet the requirements of professional traders and reducing the initial effort for anyone wishing to pursue this activity.

In particular, we will analyze and integrate possible data sources to cover the American and European market; define efficient data structures to store short-term information and end-of-day data history for long periods; develop simple and intuitive user interfaces that allow the user to perform different technical analysis with those markets.

Contents

List of Figures 4

List of Tables 6

1 Introduction 1
1.1 Overview of the Thesis . . . 1

2 Trading 3
2.1 Exchanges . . . 3
2.2 Markets . . . 4
2.3 Instruments . . . 5
2.3.1 Derivatives . . . 6
2.4 Futures . . . 7
2.4.1 Payoff . . . 8
2.5 Options . . . 8
2.5.1 Price . . . 10
2.5.2 Payoff . . . 10
2.5.3 Spread . . . 14
2.5.4 Greeks . . . 25
2.6 Platforms . . . 27
2.6.1 Competitors . . . 27

3 Essentials 33
3.1 Requirements . . . 33
3.1.1 Stakeholders . . . 33
3.1.2 Functional and Non-functional . . . 34
3.2 Data sources . . . 37
3.2.1 Types of data sources . . . 37
3.2.2 Markets web services . . . 37
3.3 Architecture . . . 40
3.3.1 Client and Server . . . 41
3.3.2 Server-side containerization . . . 42

3.4 Technologies . . . 43
3.4.1 Django . . . 43
3.4.2 React . . . 46
3.4.3 MongoDB . . . 48
3.4.4 Celery . . . 50
3.4.5 Docker . . . 50
3.4.6 Nginx . . . 52

4 Implementation 53
4.1 Database . . . 53
4.1.1 Collections . . . 53
4.1.2 Data analysis . . . 58
4.2 Async tasks . . . 60
4.2.1 update_cboe . . . 61
4.2.2 update_cme . . . 62
4.2.3 update_eurex . . . 63
4.2.4 update_history . . . 63
4.3 REST APIs . . . 64
4.3.1 Security . . . 65
4.3.2 User APIs . . . 66
4.3.3 Market APIs . . . 68
4.3.4 Chain APIs . . . 69

5 Client application 71
5.1 Data management . . . 71
5.1.1 Data store . . . 71
5.1.2 Data flow . . . 72
5.2 User Interface . . . 73
5.2.1 Markets . . . 73
5.2.2 Strategies . . . 76
5.2.3 Portfolio . . . 77
5.3 Use cases . . . 78

6 Conclusions and future works 83

List of Figures

2.1 Long and short positions of a future contract (e.g. strike price 100) 9
2.2 Long positions for call and put options contract (e.g. strike price 100 and 40 days to maturity) . . . 13
2.3 Short positions for call and put options contract (e.g. strike price 100 and 40 days to maturity) . . . 14
2.4 Bear and bull option spreads for call contracts (e.g. strike price 100 and 40 days to maturity) . . . 19
2.5 Bear and bull option spreads for put contracts (e.g. strike price 100 and 40 days to maturity) . . . 21
2.6 Long and short straddle option spreads (e.g. strike price 100 and 40 days to maturity) . . . 22
2.7 Butterfly option spreads (e.g. strike price 100 and 40 days to maturity) 24
2.8 Calendar option spreads (e.g. strike price 100 and 40 to 160 days to maturity) . . . 26
2.9 Underlying section on FiutoBeta . . . 28
2.10 Grid view in OptionRuler . . . 29
2.11 Option matrix feature in OptionVue . . . 31

3.1 Client-server architecture . . . 41
3.2 Docker compose architecture . . . 43
3.3 Django MVT Pattern . . . 44
3.4 Redux architectural pattern . . . 47
3.5 Docker client-server architecture available on this **link**. . . . 51

4.1 Historical data analysis . . . 62
4.2 Update CBOE async task . . . 64
4.3 Update CME async task . . . 64
4.4 Update EUREX async task . . . 65

5.1 Data flow example . . . 74
5.2 Markets interface . . . 74
5.3 Markets charts interface . . . 75
5.4 Markets positions interface . . . 76

5.5 Markets futures interface 76
5.6 Markets chains interface 77
5.7 Strategies interface 77
5.8 Portfolio interface 78
5.9 Creation of a new strategy inside the platform 79
5.10 Implement a butterfly spread strategy 81

List of Tables

2.1 Option chain[1] example of Eurostoxx50, with underlying price equal to 3000 and 40 days at expiration. 11
2.2 Bull call spread example, with a stop-loss equal to -3.1 and take profit equal to 6.9 16
2.3 Bear call spread example, with a take-profit equal to 6.8 and stop-loss equal to -3.2 17
2.4 Bull put spread example, with a take-profit equal to 6.89 and stop-loss equal to -3.11 17
2.5 Bear put spread example, with a stop-loss equal to -3.18 and take profit equal to 6.82 18
2.6 Long straddle spread example, with a stop-loss equal to -7.92 18
2.7 Short straddle spread example, with a take-profit equal to 7.92 . . . 20
2.8 Reverse butterfly spread example, with a stop-loss equal to -6.27 and take-profit equal to 3.73 23
2.9 Butterfly spread example, with a take-profit equal to 6.27 and stop-loss equal to 3.73 23
2.10 Calendar spread example, with a take-profit equal to 7.92 25

3.1 Stakeholders 34
3.2 Functional requirements for the platform 35
3.3 Non functional requirements for the platform 36
3.4 Relation between the URL and HTTP Methods in REST API architecture 42

4.1 User document description 54
4.2 Market document description 55
4.3 Market expiration document description 55
4.4 Chain document description 56
4.5 Strike document description 56
4.6 Option document description 57
4.7 Future document description 58
4.8 Group document description 58

4.9 Strategy document description 59
4.10 Position document description 60
4.11 Portfolio document description 61
4.12 Market history document description 61
4.13 Chain history document description 62
4.14 Historical data: collection total size in MB 63
4.15 Historical data: average document size in MB 63
4.16 Schedule time for async tasks 65

5.1 Use case: create a new strategy 80
5.2 Use case: implement a butterfly spread strategy 82

Chapter 1

Introduction

Over time, banks and online brokers have equipped themselves with valid trading systems that allow orders to be sent quickly to the market and to carry out basic chart and technical indicators. These systems, however, should be used above all for executive purposes, while the analysis and forecasting part should be carried out using appropriate external platforms. These platforms are more difficult to find especially in the field of derivatives and options and also presenting high costs making it difficult for new users to study and start to use them. Furthermore, most of these platforms are old-time software presenting portability problems with new operating systems, outdated graphical interfaces and not at all user friendly. The objective of the thesis is therefore to develop a platform capable of providing all the necessary tools to analyze the markets and show statistical results on its trend for both professional and novice traders. This platform must be usable via the web in such a way as to make it available for all operating systems present to date without the need for complicated installations.

1.1 Overview of the Thesis

In this document, will be presented all the fundamental arguments and design choices to achieve the set goal. In particular, with the following Chapter 2 we will provide a greater context on the financial world dealing with the main suppliers of futures and options contracts such as CME and CBOE for the American markets and EUREX for the European one to then deepen the concept of derivative instruments. With regard to this, we will describe in detail what futures and options contracts are, showing their characteristics and context of use with the relative benefits and the possible operational strategies that can be implemented. This step is important to provide the right context information and to better understand the functional requirements requested by the user.

Then we move on to the practical act in the Chapter 3 by analyzing and collecting the user requirements in order to identify and classify the most important ones for the first version of the application. This information are essential for understanding what data we need, where to retrieve them and how often their needs updating. It was therefore possible to proceed with the search for data sources capable of satisfying these requirements, which ended with the use of the main providers mentioned above. After that, we moved on to the next step concerning the most appropriate design and architecture for the development of the system and closing with the discussion of the architecture and technologies chosen for it.

In Chapter 4 we focused on the implementation of the application. The first step was to define the entities that best represent our financial concepts and relationships that best model the user operations. From a practical point of view it translates into the definition of models and collections for the non-relational database. Then we moved on to the development of asynchronous tasks with the purposes of contacting the external data sources and successively mapping their information in the internal models and then saving them in the database. Finally, various web services were described to allow the user to view these financial information but also to process and store their operational choices within the available markets.

Finally, the Chapter 5 deals with the client application to be used by the end user. Here we will show how the information received from dedicated web services are stored and displayed and how requests and events coming from the interaction with the user are managed. Then they will be showed section by section all functionalities available for the user. Furthermore, some typical scenarios and use cases by the user will be exposed during the normal course of his trading activity.

Chapter 2

Trading

Trading is the activity of buying and selling financial instruments such as shares, options, currencies and futures and whose value is listed during the opening phases of the world exchanges. Trading success requires the mastery of many subjects, from strategies to instruments and depends on a trader's ability to be profitable over time. Implies active participation in the markets as opposed to investing, which suggests a buy-and-hold strategy.

In this chapter we will initially try to provide an overview of the components and main players that are part of the trading and then conceptually provide deepen knowledge of the financial instruments that are present within the platform.

2.1 Exchanges

An exchange is an open and organised marketplace where financial instruments such as securities, commodities, derivatives and other are grouped in their related financial markets and traded. Exchanges have the responsibility to provide equitable, well-ordered and efficient trading and spread of price information by centralising the buying and selling of a particular asset and ensure that all trades are executed at the best available market price. For our purposes we want to focus in particular on the following exchanges: CBOE, CME and EUREX.

Cboe Options Exchange was founded in 1973 and become the world's largest options exchange [6]. Provide contracts for several financial instruments and products like individual equities, indexes, and interest rates, call and put options, thousands of publicly traded stocks, as well as on exchange-traded funds (ETFs) and exchange-traded notes (ETNs). Finally, the Cboe Volatility Index (VIX) index, which is the premier barometer of equity market volatility. This Index is based on real-time prices of near-the-money options on the S&P 500 Index (SPX) and is designed to reflect investors' consensus view of future (30-day) expected stock

market volatility.

Chicago Mercantile Exchange is an organized exchange founded in 1898 and colloquially known as the Chicago Merc [7]. Provide several contracts for the trading of futures and options in the sectors of agriculture, energy, stock indices, foreign exchange, interest rates, metals, real estate. It have even recently introduced Bitcoin futures and weather derivatives.

Eurex Exchange was founded in 1998 and become one of the largest futures and options markets in the world [8]. Provide products that range from German and Swiss debt instruments to. It allows traders to operate online and despite being based in Europe thanks to their electronic access is available from 700 locations around the world. Nowdays, is part of the Eurex Group and is owned by Deutsche Börse AG, a transaction services provider that enables access to global capital markets to investors and financial institutions.

2.2 Markets

As we already said, exchanges contain different types of markets enabling trading of financial instruments, such as stock, bond, forex, derivatives asset and many more. In this section a brief introduction is given about the principal market categories describing the financial instruments they group, which are the main actors that operate and the purpose of their use.

Stock Markets where shares or common stocks are allowed to buy and sell, and enable the subsequent trading thereof. Most common participants in this market are investors and traders, both retail and institutional, as well as market makers and specialists who maintain liquidity and provide two-sided markets. Instead, there are some third parties, like brokers, that facilitate trades between buyers and sellers without take a personal position in a stock.

Index Markets where more than one share is grouped together and the value of a single index thus represents all the underlying shares. This value can be calculated in three different ways:

- equally weighted, where all securities have the same weight within the index
- price weighted, where the value depends on the sum of the prices of the Stocks that compose it (the more expensive Stocks have a greater weight)
- value weighted, where the weight of each security is proportional to its market capitalization.

Most of the world's major indices are calculated using the value weighted methodology, for example S&P 500 and DAX. While the Dow Jones Industrials expresses itself through the price weighted method.

Bond Markets also referred as debt, credit, or fixed-income market where corporations and governments issue bond securities to finance projects and operations. Those securities are used by investors to loans money for a defined period with a pre-established interest rate.

Money Markets where short term debt financing and investment are provided and involve large-volume trades between institutions and traders. Individuals may also invest by buying short-term certificates of deposit (CDs), banker's acceptances, certain bills, notes and commercial papers. This market is characterized by a high degree of safety and a relatively low return in interest.

Derivatives Markets where instruments that derive their value from other underlying instruments like bonds, indexes, stocks and other, can be bought or sold. Generally those instruments are futures and options contracts used by market participants for speculate on the price movement of the underlying without physically owning it.

Forex Market where exchange rates between currency pairs can be used as speculation instruments by the participants. Is the most liquid market, decentralized and composed of a global network of computers and brokers and made up of banks, investment management firms, and investors.

Commodities Markets where physical goods, such as agricultural products, energy products or precious metals are exchanged for money, between producers and consumers.

Cryptocurrency Markets where digital currencies can be traded such as Bitcoin and Ethereum, is decentralized without the need for an actual exchange authority to facilitate the transactions.

2.3 Instruments

In the financial literature the concept of financial instruments indicates any assets or contract that can be purchased, traded, created, modified, or settled for. In terms of contracts, there is a legal agreement between involved parties during a financial instrument transaction. Financial instruments may be divided into three types: cash instruments, foreign exchange instruments, and derivative instruments.

Cash Instruments are financial instruments where their values reflecting markets condition and are divided in two types; securities and deposits, and loans. In particular, securities are a financial instrument that represents ownership of a company that is publicly-traded. Those instruments have a monetary value and are traded on the stock market. Deposits and Loans are monetary assets with contractual agreement between parties and for this reason are considered cash instruments.

Foreign Exchange Instruments are currency agreements that are represented on the foreign market and they can be divided into three categories: spot, outright forwards and currency swap. The first one, are agreements with limited amount of time, at most two days, for performing the exchange. Outright forwards, are agreements that allows to perform the exchange of currency "forwardly" but before the agreed date. Finally, currency swap are agreements based on buying and selling currencies with different dates simultaneously.

Derivative Instruments are financial instruments with a price that is not autonomous but derives from one or more underlying assets and the price value is determined by fluctuations of it. The most common underlying assets include stocks, bonds, commodities, currencies, interest rates and market indexes.

2.3.1 Derivatives

The origin of derivative instruments stems from the trading of raw materials and in particular from the need to protect the delivery of goods and the main function was typically to cover the risk of any unexpected fluctuation in prices due to various causes. Over time derivatives have mostly changed this function and now are used for speculation. In particular, can be used to hedge a position and speculate on the directional movement of an underlying asset. Furthermore, the total required capital to operate with derivatives is lower with respect to other financial markets. Indeed, with these financial instruments it is possible to invest only a part of the total value of the underlying asset, which normally corresponds to about 10%. The five most common examples of derivatives instruments are:

- Synthetic Agreement for Foreign Exchange (SAFE) are contracts that guarantees for a period of time a specified exchange rate in the over-the-counter (OTC) market.
- Forward are contracts where at expiration two parties undertakes an exchange of derivatives at a specific price.
- Futures are contracts where two parties undertakes an exchange of derivatives on a determined future date and at a forward exchange price.

- Options are contracts where two parties, seller and buyer, undertakes an exchange of a specific quantity of derivatives and the seller grants the buyer the right to buy or sell them at a predetermined price within the expiration date.
- Interest Rate Swap are contracts where each party agrees to pay other interest rates on their loans in different currencies.

For our purposes, we will discuss Futures and Options more in detail in the next sections.

Definitions

Before going into detail with derivatives instruments it is important to have in mind some basic metrics and definitions:

- Payoff is a chart that represents the profit trend in the event that a new position is opened. On the horizontal axis there are all the possible prices of the underlying asset while on the vertical axis there are the possible profits or losses.
- Fair value, generally, represents the theoretical value of the underlying compared with its market prices and verifies whether the prices are overestimating or underestimating the asset or if the value itself is in equilibrium.
- Volumes are the number of transactions carried out in a given period of time and indicate the liquidity of a given financial asset. Higher volumes mean greater liquidity. Few volumes means that the title is not very liquid and little treated.
- Open interest represents the precise number of contracts left on the market and can increase or decrease with the opening or closing of positions. It is particularly important and there are some advanced techniques for reading the market which cannot be applied without this data.

2.4 Futures

Futures are an agreement between two parties in which they trade a quantity of an underlying asset, at a forward price and at an expiration date. Both parties are obligated to fulfill a commitment to buy or sell the underlying asset. Indeed, the buyer of the future contract opens a **long** position and committed to pay a price at a certain date to receive the underlying asset. The seller opens a **short** position and undertakes to deliver the underlying asset in exchange for a price, at

an expiration date. If in the meantime the price of the underlying rises, the one who bought the future makes a profit, while the one who sold the future takes a loss. Clearly, it is the exact opposite situation if the price of the underlying asset falls because this kind of contracts are zero-sum. Furthermore, it is important to add that futures have quarterly maturities: March, June, September and December and in financial systems these maturities are coded with the following letters: H, M, U, Z.

By the way a derivative contract, written on a financial asset, is almost never used with the intent of concluding the transaction with the physical delivery of the underlying asset. In reality, only 3% of contracts are completed because in the majority of cases are used for speculation.

2.4.1 Payoff

The term payoff indicates the yield of a single futures contract currently being traded. By plotting the respective charts for the long and short position (Figure 2.1) it is possible to notice two completely different profit trends. In particular, when a long position is opened the profit tend to go up as the price of the underlying increases, vice versa with a short position the profit go down if the price of the underlying goes down. The cross with the horizontal axis indicates the price of the underlying when the position was opened.

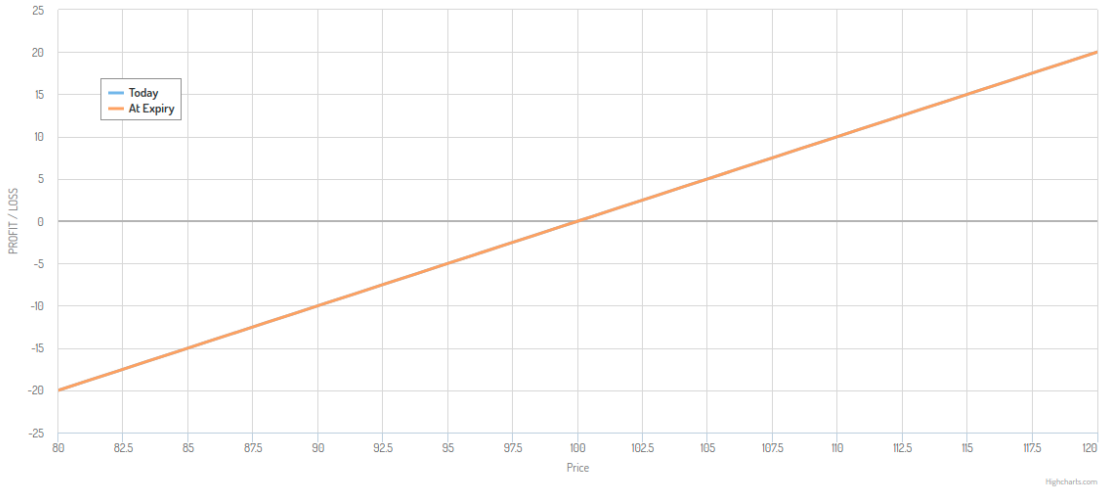
Thus, the main characteristic of futures contracts is the possibility to have a linear dependence on the price of the underlying without actually paying its full value. In fact, to open a future position it is necessary to pay an initial cost of the so-called guarantee margin equal to approximately 10% of the real value of the underlying. This characteristic is called the leverage effect and can be a source of large profits but also of large losses if not used correctly.

2.5 Options

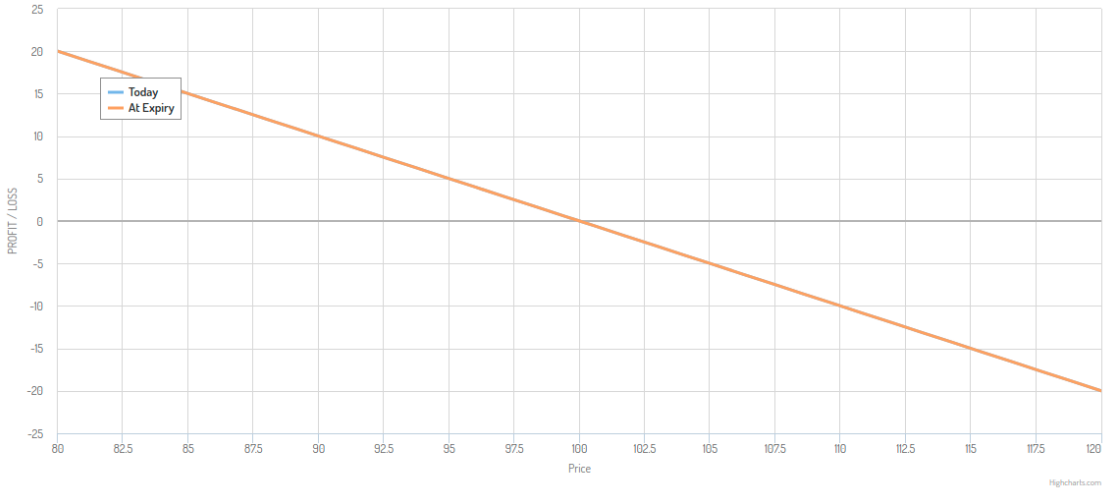
An options contract is similar to a futures one with the key difference that the buyer have the right, but not the obligation, to buy or sell a given amount of a particular underlying asset at a predetermined price, commonly called strike price [1], somewhere in the future. The seller, on the other hand, earn the prize, and has the obligation to collect or deliver the underlying at the will of the buyer.

Furthermore, options allow to build more advanced speculative strategies for all underlying assets with the possibility to change the risk profile at any time. However, options are particularly more complex and do not have the linear profit and loss payoff of the future contract. They are subject to multiple variables that

[1] the price at which a put or call option can be exercised



(a) Long future



(b) Short future

Figure 2.1: Long and short positions of a future contract (e.g. strike price 100)

modify their price and leverage. For this reason before going into detail on the most common operational strategies it is important to have a look of some definitions and math prerequisites.

Type of options options are divided in two types: call and put options. Call options are derivative instruments that guarantees the buyer the right, but not the obligation, to buy an underlying at a predetermined price at maturity. Put options are a derivative instruments that guarantees the buyer the right, but not the obligation, to sell the underlying at a predetermined price at maturity. Furthermore,

there is another important geographical division: in the case of European options, it is possible to exercise one's right only upon expiry, while American options give the holder the possibility of exercising at any time within the expiry date. This wider possibility for American options means that their theoretical price is higher than that of similar European options (for options with the same duration, the same underlying and the same strike), and in any case never lower.

2.5.1 Price

The price of an option contract depends on many factors. In particular, the option pricing formula is based on a model of three mathematicians-economists Black, Sholes and Merton, more simply called the Black and Sholes model, who were awarded the Nobel Prize for this theorization. This model tells us that the factors that affect the price of an option are: price of the underlying, strike price, expiration date, interest rate, dividends and volatility. Among these the strike price, or the exercise price of the option, and the relative distance with the underlying price plays an important role. This distance represents the **intrinsic value** component and divide each option type in three price ranges (Table 2.1):

- At the money (ATM) when a put or a call option has the strike price equal to or close to the underlying price.
- In the money (ITM) when a call option has the strike price lower or a put option has the strike price higher then the underlying price.
- Out the money (OTM) when a call option has the strike price higher or a put option has the strike price lower then the underlying price.

The price of an option has also an high dependence with the **volatility** that, within the number of days left to the expiration, compose the **time value** component. Volatility is difficult to quantify, represents the measure of the price movements of the underlying. Large price fluctuations lead to increased volatility with a high probability that a certain option will go from ATM/OTM to ITM, while small swings lead to low volatility with a high probability that options will expire worthless. To measure it, statistical tools such as standard deviation are used.

Finally, these two components affect the price based on the price range the option is in. Indeed, the ITM option price depend on the intrinsic value and time value, while the ATM and OTM options price depend only on time value.

2.5.2 Payoff

As already mentioned above, in the case of options, the payoff has no longer a linear dependence on the underlying. In particular, due to their complexity from a

State	Call	Strike	Put	State
ITM	219	2850	69	OTM
ITM	201	2875	76	OTM
ITM	183	2900	83	OTM
ITM	166	2925	91	OTM
ITM	150	2950	100	OTM
ATM	134	2975	109	ATM
ATM	**120**	**3000**	**120**	**ATM**
ATM	107	3025	132	ATM
OTM	95	3050	145	ITM
OTM	83	3075	158	ITM
OTM	73	3100	173	ITM
OTM	64	3125	189	ITM
OTM	56	3150	206	ITM

Table 2.1: Option chain[1] example of Eurostoxx50, with underlying price equal to 3000 and 40 days at expiration.

[1] Set of options with the same expiration date

mathematical point of view, two important payoff must be taken into consideration: "at now" and "at maturity". The payoff at maturity, as in futures, shows what the profit will be on the contract expiration date. While the payoff at now, present only for options, shows the trend of the profit in each instant, that is, the gain or loss compared to the option price in each instant. The distance between those two lines is defined as a time value. Over time, this value will decrease and consequently the at now line will get closer and closer to the pay off at maturity until it coincides with it on the day of maturity. Another important concept to consider is the division of options into call and put. In fact, starting from this simple division, we can perform four basic positions that can be opened: Long Call/Put and Short Call/Put.

Long Call

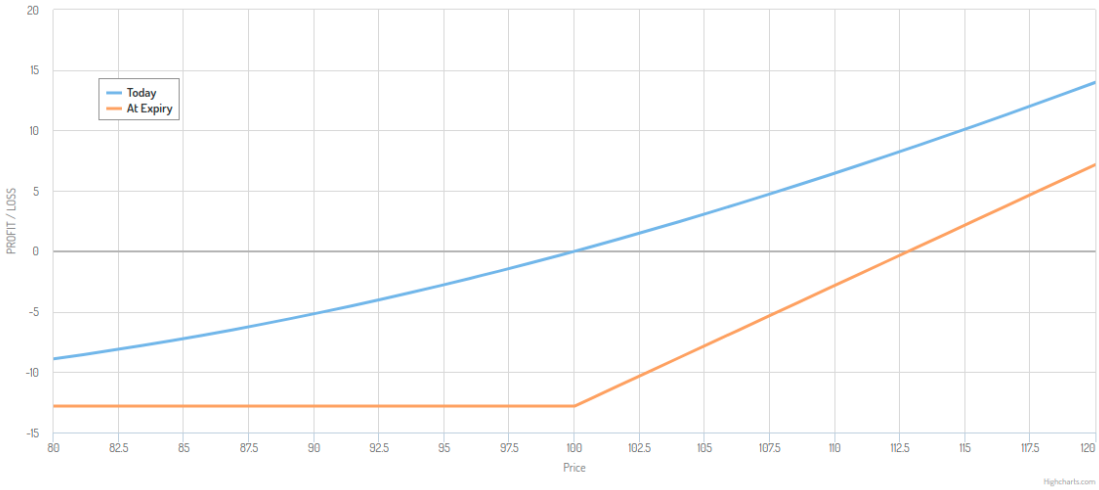
It is a strategy that involves buying a call option to obtain the right to buy the underlying at a predetermined strike within a certain expiry date; So there is an initial cost, also called a prize, which is reflected in the graph (Figure 2.2a) with the payoffs which, at the value of the strike price, start from a negative profit. It is clear that those who execute a long call strategy believe that the price of the underlying can rise in the future and then exercise the right to buy it at a lower predetermined price and resell it immediately to close with a positive profit.

Long Put

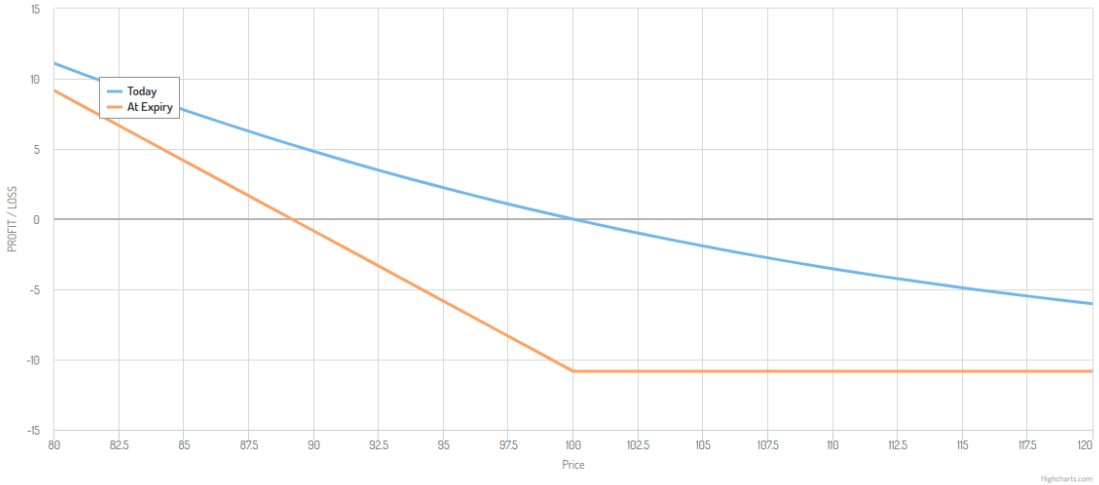
Here, instead, the strategy involves buying a put option to obtain the right to sell the underlying at a predetermined strike within a certain expiry date; The payment of an initial cost, also called a prize, is reflected in the graph (Figure 2.2b) with the payoffs which, at the value of the strike price, start from a negative profit. It is clear that those who execute a long put strategy believe that the price of the underlying may fall in the future and then exercise the right to sell it at a higher predetermined price by closing in positive.

Short Call

It is a strategy that involves selling a call option to leave the possibility to other parties, the right to buy the underlying at a predetermined strike within a certain expiry date. By selling, you get the prize that was paid by whoever bought the respective call. For this reason, in the graph (Figure 2.3a) the payoffs, at the strike value, start from a positive profit. It is clear that those who execute this strategy believe that the price of the underlying cannot rise above the predetermined strike value so as not to lose the prize received and close in positive.



(a) Long Call

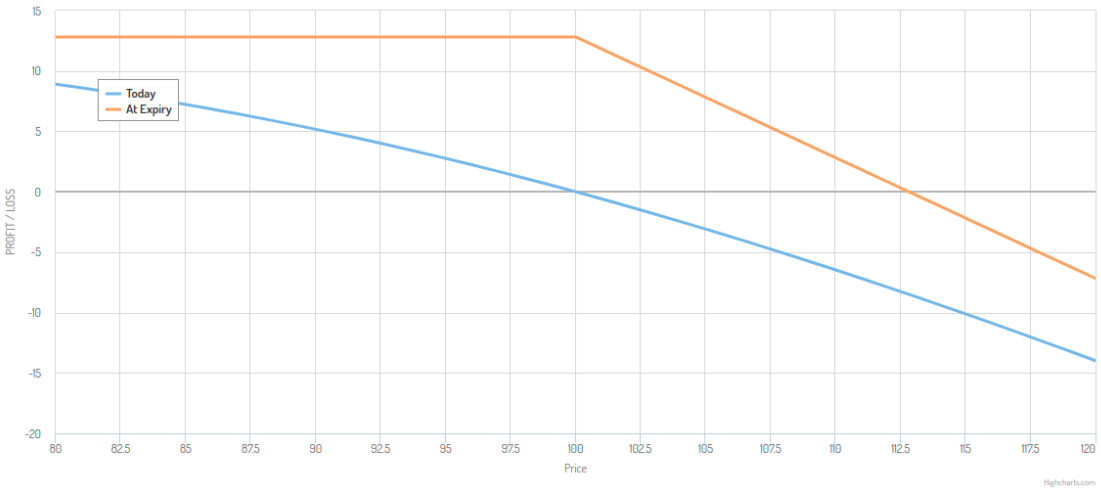


(b) Long Put

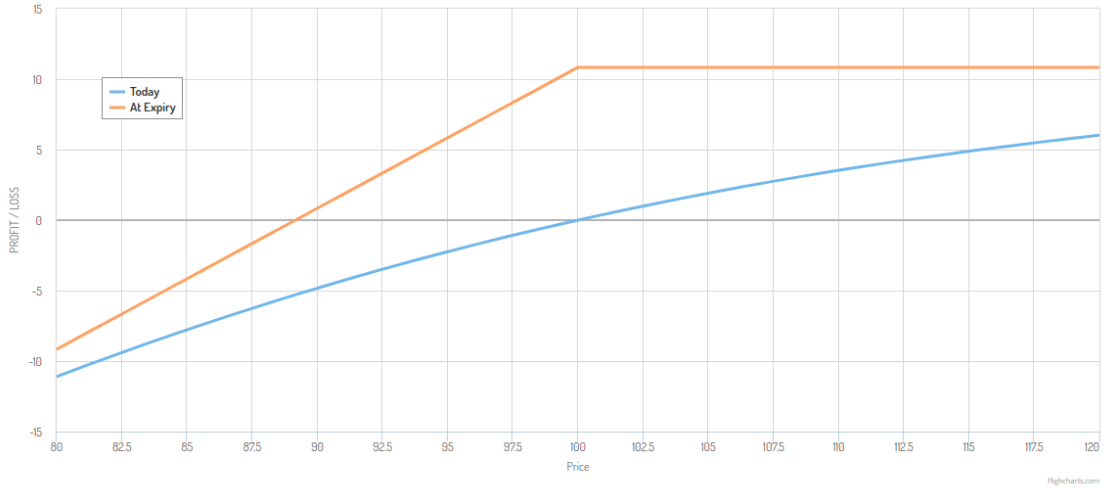
Figure 2.2: Long positions for call and put options contract (e.g. strike price 100 and 40 days to maturity)

Short Put

It is a strategy that involves selling a put option to leave the possibility to other parties, the right to sell the underlying at a predetermined strike within a certain expiry date. By selling, you collect the prize that was paid by whoever buys the put. For this reason, in the graph (Figure 2.3b) the payoffs, at the strike value, start with a positive profit. It is clear that those who execute this believe that the price of the underlying cannot fall below the pre-established strike value so as not to lose the prize received and close in positive.



(a) Short Call



(b) Short Put

Figure 2.3: Short positions for call and put options contract (e.g. strike price 100 and 40 days to maturity)

2.5.3 Spread

The option spread is a strategy that involves the combined use of two or more basic buy and sell positions, in order to build a risk profile consistent with the market and with proper risk management expectations. With the spreads, is possible to modulate the trade off between being a seller and therefore collecting a small prize for a significant number of winning trades but facing significant risks, and being a buyer and therefore paying a small prize with the possibility of ride a big movement with big gain. The three main categories of spreads are:

- vertical spread, consisting of options with the same maturity but on a different strike price
- horizontal spread, consisting of options with different maturities but on the same strike price
- diagonal spread, consisting of options with different strikes and maturities

Vertical spreads focus on the directionality of trading. While the horizontal ones benefit from the different time decay and volatility of the different maturities. Finally, diagonal spreads are useful for benefiting from both directionality and different volatility and time decay. Moreover, based on the difference between prize paid and received, the spreads can be classified into:

- debit spread, when the prize received by selling options is lower than the prize paid for purchasing options
- credit spread, when the prize received by selling options is higher than the prize paid for purchasing options

Option spreads are one of the best ways to work with options. The total operating costs is reduced by collecting prizes through the sale of some contracts and is possible to obtain strategies with well-defined stop loss and take profit. Some strategies that can be achieved through option spreads will be discussed in this section, further highlighting their advantages and disadvantages.

Bull Call

The Bull Call spread is a debt, vertical option spread strategy which can be obtained by purchasing a call option at a lower strike price and selling another call option at a higher strike within the same expiration date (Table 2.2). The difference between the prize received and the one paid represents the total cost of the operation also called stop-loss[2]. However, there is also a take profit[3] given by the difference between spread[4] and the total cost. It is also possible to understand that the profit-loss ratio is greater than one, in fact the area above zero is greater the below one (Figure 2.4a).

This strategy is useful when we want to exploit a possible increase in prices, perhaps even reaching the strike of the option sold.

[2] pre-established risk limit

[3] maximum earning limit

[4] the difference in strike price between the option sold and the one purchased

Bull Call Spread					
Operation	**Quantity**	**Option**	**Strike**	**Expiration Date**	**Prize**
Buy	1	Call	100	Sep 30	-4.07
Sell	1	Call	110	Sep 30	0.97

Table 2.2: Bull call spread example, with a stop-loss equal to -3.1 and take profit equal to 6.9

Bear Call

The Bear Call spread is a credit, vertical spread strategy which can be obtained by purchasing a call option at a higher strike price and selling another call option at a lower strike within the same expiration date (Table 2.3). In this way the definitions of stop-loss and take-profit are reversed with respect to the bull call spread. Indeed, the difference between the prize received and the one paid represents the take-profit while the stop loss is given by the difference between the spread[4] and the take-profit itself.

Bear Call Spread					
Operation	**Quantity**	**Option**	**Strike**	**Expiration Date**	**Prize**
Buy	1	Call	100	Sep 30	-4.07
Sell	1	Call	90	Sep 30	10.87

Table 2.3: Bear call spread example, with a take-profit equal to 6.8 and stop-loss equal to -3.2

This strategy (Figure 2.4b) is very useful for operating on a market on which we expect an unchanged or a decrease of the prices, possibly with a movement that reach beyond the strike of the option sold.

Bull Put

The Bull Put spread is a credit, vertical spread strategy which can be obtained by purchasing a put option at a lower strike price and selling another put option at a higher strike within the same expiration date (Table 2.4). The difference between the prize received and the one paid represents the maximum take-profit that will have from these operations. Instead the stop-loss is given by the difference between the spread[4] and the take-profit. This strategy is useful when we want to exploit

Bull Put Spread					
Operation	**Quantity**	**Option**	**Strike**	**Expiration Date**	**Prize**
Buy	1	Put	100	Sep 30	-3.86
Sell	1	Put	110	Sep 30	10.73

Table 2.4: Bull put spread example, with a take-profit equal to 6.89 and stop-loss equal to -3.11

an unchanged or possible increase of the prices, perhaps even reaching the strike of the option sold.

Bear Put

The Bear Put spread is a debt, vertical spread strategy which can be obtained by purchasing a put option with a higher strike and selling another put option with a lower strike within the same expiration date (Table 2.5). In this way the definitions of stop-loss and take-profit are reversed with respect to the bull call spread.

Bear Put Spread					
Operation	**Quantity**	**Option**	**Strike**	**Expiration Date**	**Prize**
Buy	1	Put	100	Sep 30	-3.85
Sell	1	Put	90	Sep 30	0.67

Table 2.5: Bear put spread example, with a stop-loss equal to -3.18 and take profit equal to 6.82

Looking the payoff (Figure 2.5b) is possible to see that the substantial difference lies in the direction of the price. In fact, this strategy is useful for taking advantage of a decrease in prices possibly with a movement that reaches the strike of the option sold.

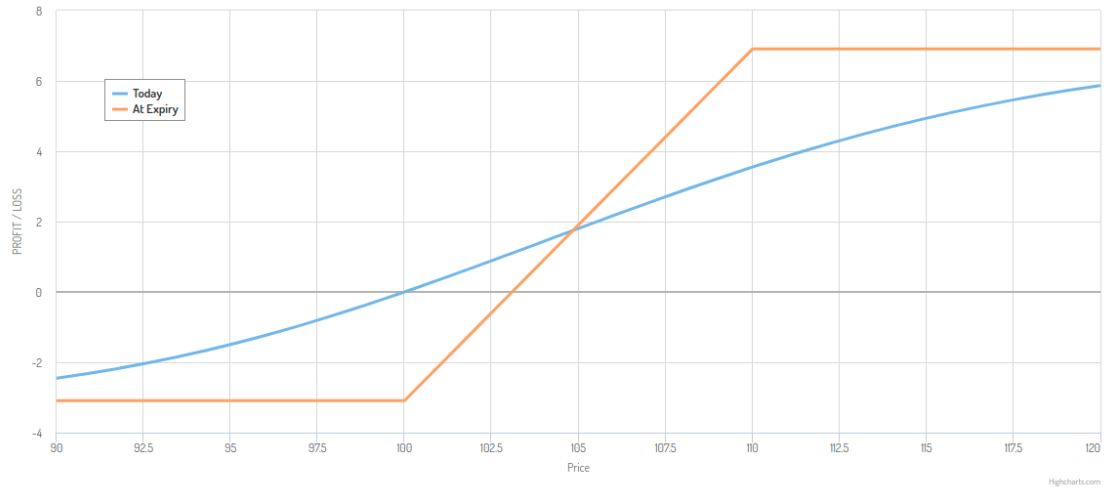
Long Straddle

The Long Straddle spread is a debt, vertical spread strategy which is obtained by purchasing an equal amount of call option and put option on the same strike price and expiration date (Table 2.6). In this case the stop-loss is defined as the sum of all the prizes paid while the take-profit is not present, so we can say that we do not have a maximum profit limit.

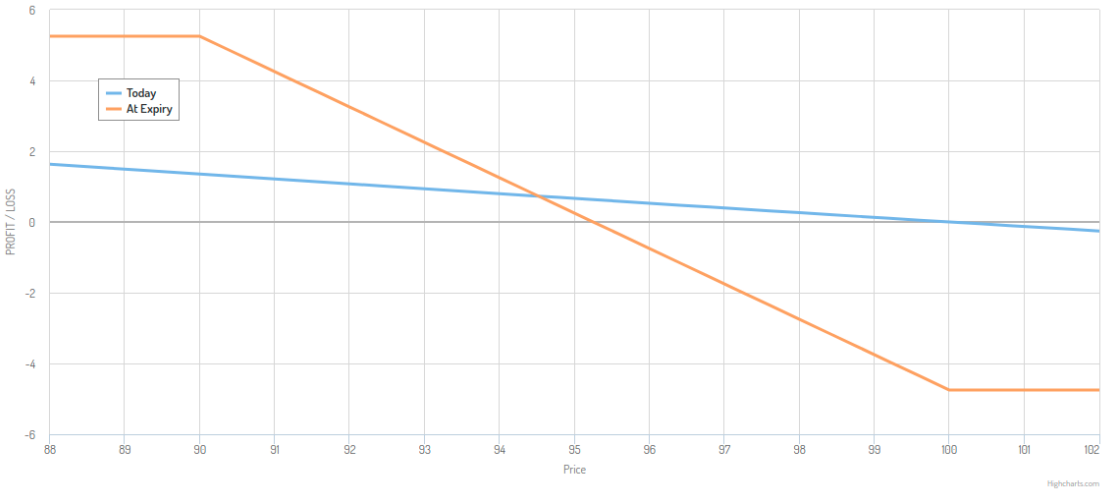
Long Straddle Spread					
Operation	**Quantity**	**Option**	**Strike**	**Expiration Date**	**Prize**
Buy	1	Call	100	Sep 30	-4.07
Buy	1	Put	100	Sep 30	-3.85

Table 2.6: Long straddle spread example, with a stop-loss equal to -7.92

Looking at the payoff chart (Figure 2.6a) is possible to notice that this strategy becomes useful when you want to exploit large movements in the price, either upwards or downwards. What matters is that the market moves quickly and deeply, no matter the direction. The main advantages are the risk limited only by the prize we paid and theoretically unlimited profit possibilities. The disadvantages are represented by the total cost of the prize required to enter the market, because we are buying two options, and if there is no strong directional movement is very likely to lose all or part of the prize.



(a) Bull Call Spread



(b) Bear Call Spread

Figure 2.4: Bear and bull option spreads for call contracts (e.g. strike price 100 and 40 days to maturity)

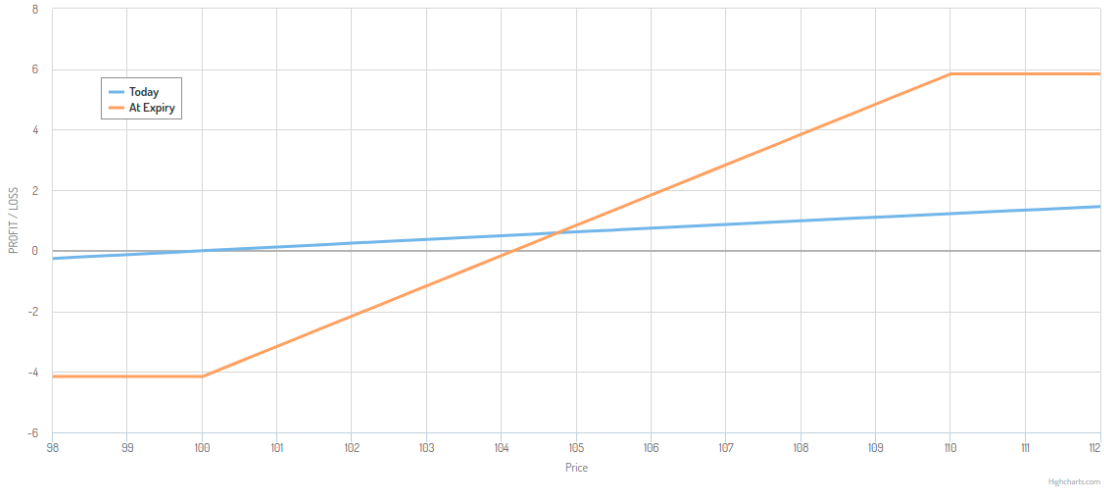
Short Straddle

The Short Straddle spread is a credit, vertical spread strategy which is obtained by selling an equal amount of call option and put option on the same strike price and expiration date (Table 2.7). In this case the, there is an inverted situation from the long straddle spread. There is a take-profit, defined as the sum of all the prizes received, but there is not a pre-established risk limit.

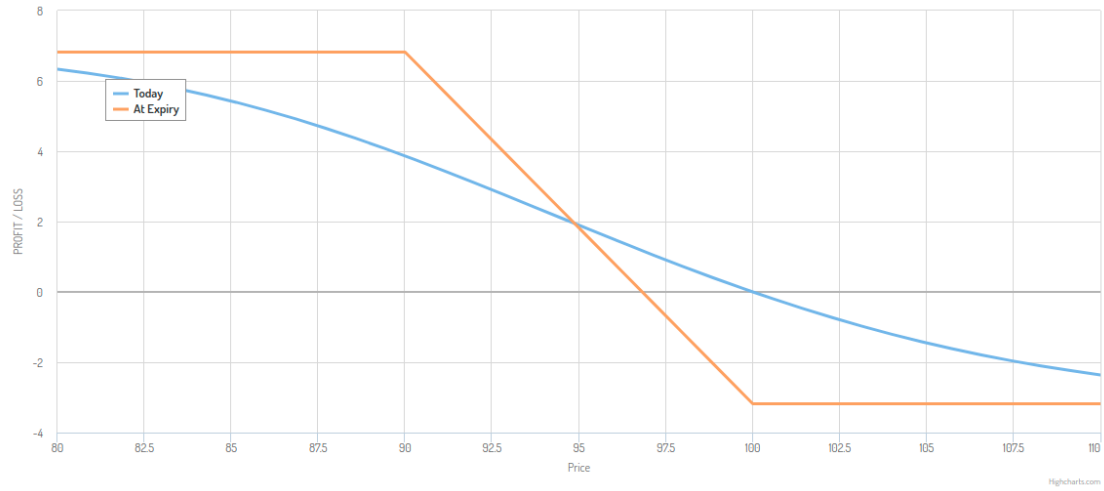
Looking at the payoff chart (Figure 2.6) is possible to notice that this strategy becomes useful when you want to exploit an unchanged situation in the prices.

Short Straddle Spread					
Operation	**Quantity**	**Option**	**Strike**	**Expiration Date**	**Prize**
Sell	1	Call	100	Sep 30	4.07
Sell	1	Put	100	Sep 30	3.85

Table 2.7: Short straddle spread example, with a take-profit equal to 7.92



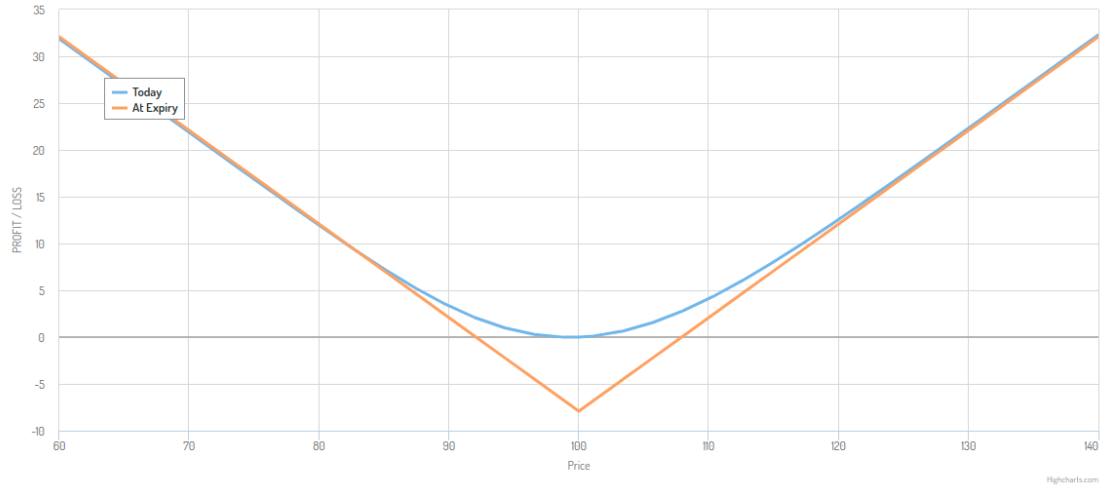
(a) Bull Put Spread



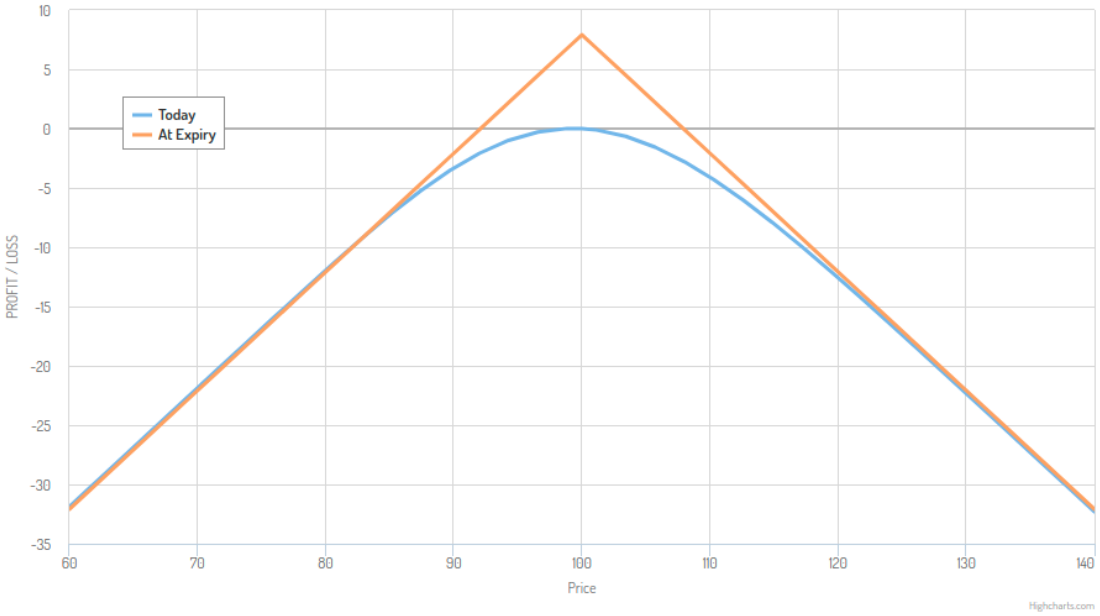
(b) Bear Put Spread

Figure 2.5: Bear and bull option spreads for put contracts (e.g. strike price 100 and 40 days to maturity)

What matters is that the market moves slowly, and in a shallow way, without directionality and statistically this behavior is much more likely. The main advantages are the low operating costs and an high probability the prices does not change too much. The disadvantage is that potential losses are high if the price changes a lot as there is no stop-loss.



(a) Long Straddle Spread



(b) Short Straddle Spread

Figure 2.6: Long and short straddle option spreads (e.g. strike price 100 and 40 days to maturity)

Reverse Butterfly

The Reverse Butterfly spread is a debt, vertical spread strategy and extends the long straddle spread strategy. In fact, in addition to buying an equal number of call and put on the same strike price you have to sell an equal number of call and put on different strikes (Table 2.8). This allows to reduce the disadvantage of having a

high operating cost, collecting the prizes obtained by the sale of the put.

Reverse Butterfly Spread					
Operation	**Quantity**	**Option**	**Strike**	**Expiration Date**	**Prize**
Buy	1	Call	100	Sep 30	-4.07
Buy	1	Put	100	Sep 30	-3.85
Sell	1	Call	110	Sep 30	0.98
Sell	1	Put	90	Sep 30	0.67

Table 2.8: Reverse butterfly spread example, with a stop-loss equal to -6.27 and take-profit equal to 3.73

Looking at the payoff chart (Figure 2.7a) is possible to notice that we can still exploit large movements in the price, either upwards or downwards with a reduced cost but with a maximum profit.

Butterfly

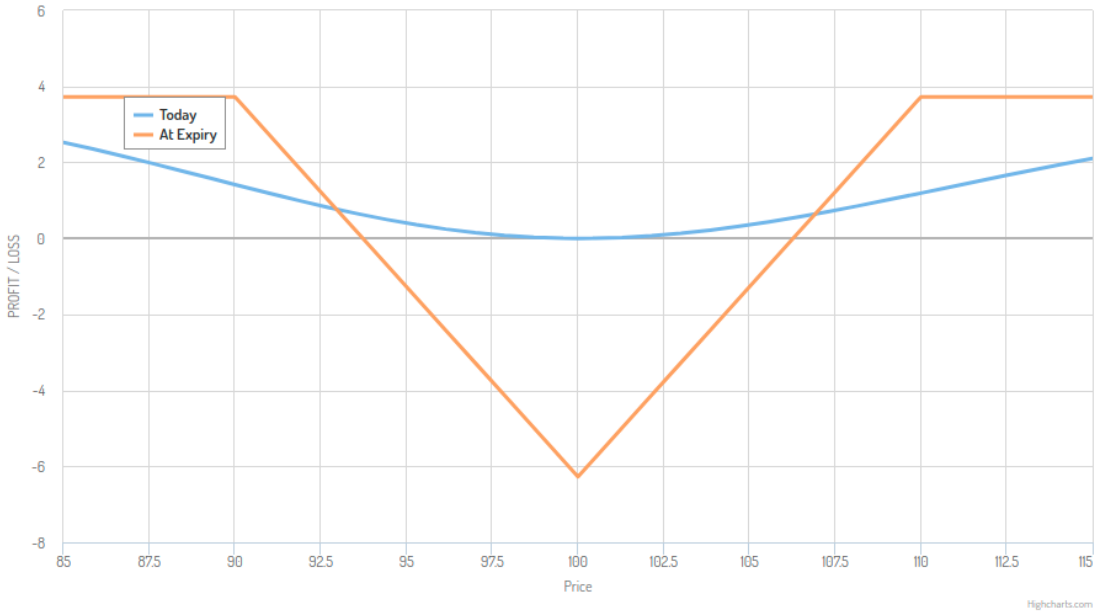
The Butterfly spread is a credit, vertical spread strategy and extends the short straddle spread strategy. In fact, in addition to selling an equal number of call and put on the same strike price you have to buy an equal number of call and put on different strikes (Table 2.9). In this way we lose some profit respect to the short straddle strategy but we have a covered portfolio thanks to the presence of a stop-loss (Figure 2.7b).

Butterfly Spread					
Operation	**Quantity**	**Option**	**Strike**	**Expiration Date**	**Prize**
Sell	1	Call	100	Sep 30	4.07
Sell	1	Put	100	Sep 30	3.85
Buy	1	Call	110	Sep 30	-0.98
Buy	1	Put	90	Sep 30	-0.67

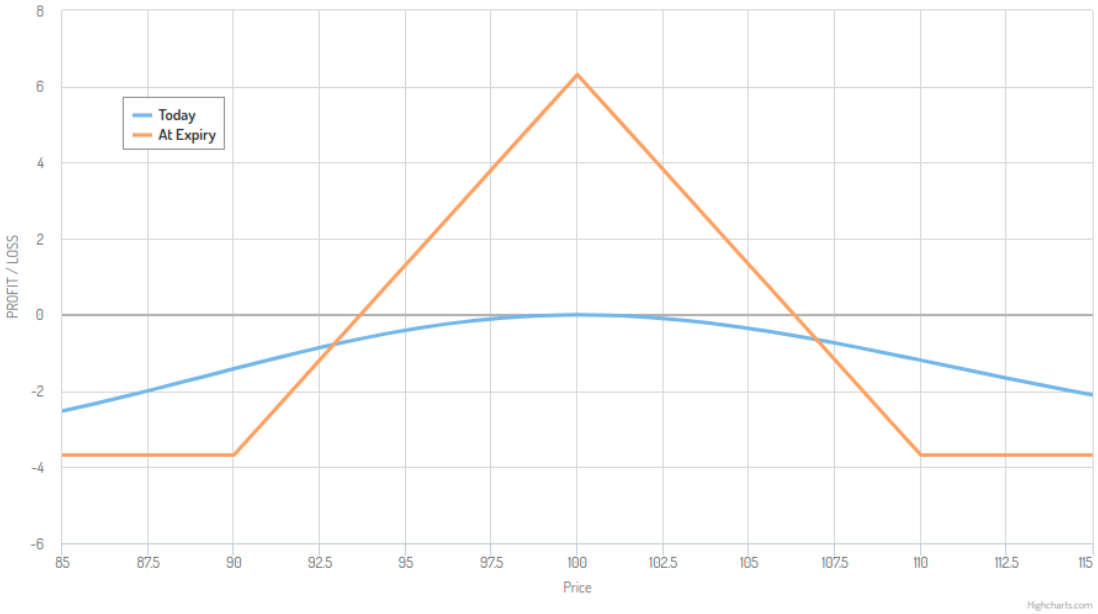
Table 2.9: Butterfly spread example, with a take-profit equal to 6.27 and stop-loss equal to 3.73

Calendar Spread

The Calendar Spread is a horizontal spread strategy obtained by buying and selling options on the same strikes, but employing different expiration dates. This kind of strategies benefits from the greater temporal decay of the closest-maturing option sold, while the far-maturing option retains its value. They may be debit or credit depending on whether the prize paid is higher or lower than the prize received and



(a) Reverse Butterfly Spread



(b) Butterfly Spread

Figure 2.7: Butterfly option spreads (e.g. strike price 100 and 40 days to maturity)

the type of option does not matter as long as they are either all put or all call (Table 2.10).

Looking at the payoffs (Figure 2.8) is possible to notice that the debit calendar spread can be applied when the volatility is low and therefore a slow fluctuation of the prices. To the contrary a credit calendar spread can be applied in order to take advantage of the possible strong fluctuations of the price both upwards and downwards.

Table 2.10: Calendar spread example, with a take-profit equal to 7.92

Type	Operation	Quantity	Option	Strike	Expiration Date	Prize
Debit	Sell	1	Call	100	Sep 30	4.07
	Buy	1	Call	100	Dec 31	-8.32
Credit	Buy	1	Call	100	Sep 30	-4.07
	Sell	1	Call	100	Dec 31	8.32

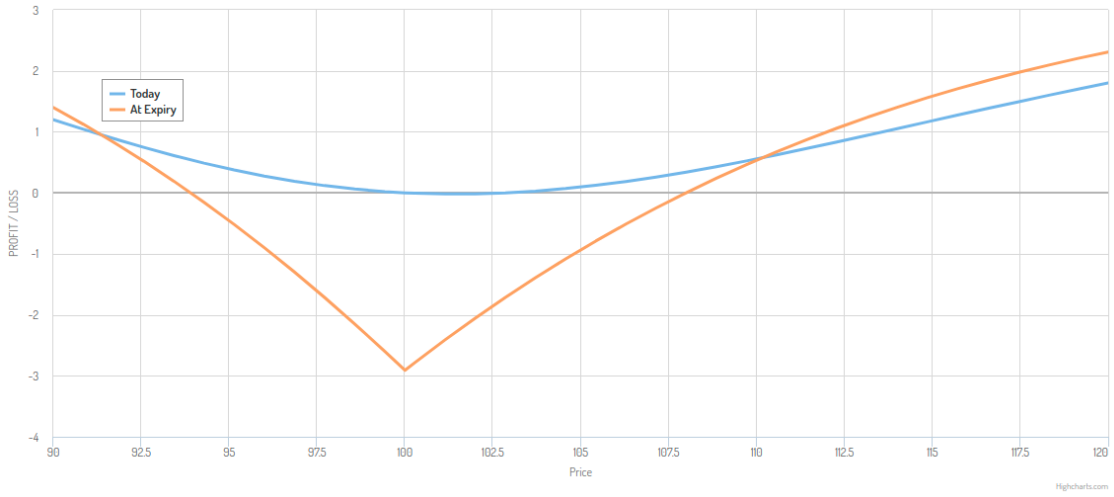
2.5.4 Greeks

Greeks are numerical values useful to understand possible operational opportunities to use for increase your profit. They measure the sensitivity of the option price to the various factors that compose it and they are expressed only for the at now payoff. The main Greeks are:

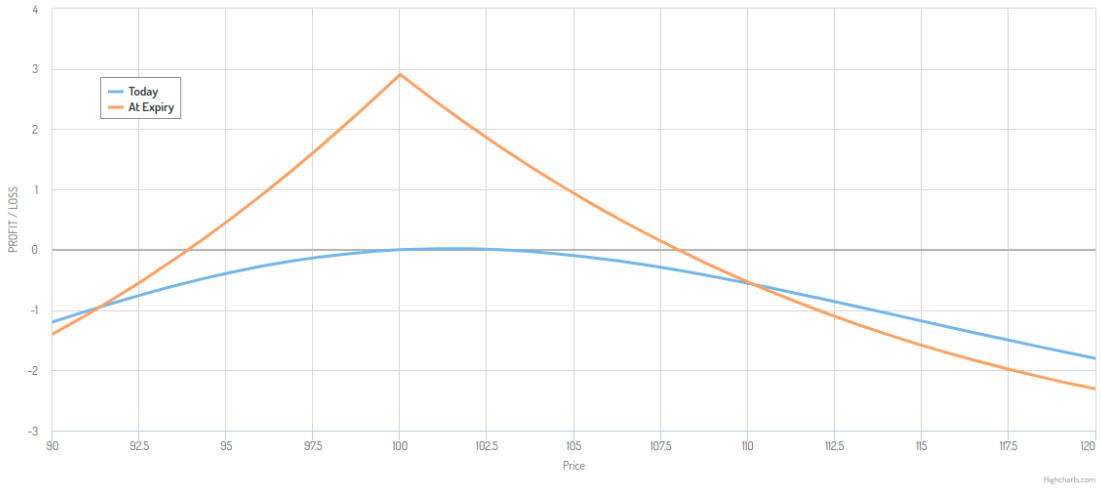
- **Delta** represents the first derivative of the option price compared to the price of the underlying asset. For both put and call instruments, the delta increases as the underlying price increases and decreases as the underlying price decreases. In particular, in the case of call options the delta always assumes a value between 0 and 1, while in the case of put options it always assumes values between -1 and 0. Furthermore, delta also represents the probability that the option will expire in the money or out of the money. Indeed, with the decrease of the days to maturity, delta tends to assume values equal to 1 or -1 for call and put ITM, equal to 0 for OTM and 0.5 for ATM.

- **Gamma** measure how much the delta varies as the underlying market changes. Generally it has the maximum value for ATM options to decrease as options become OTM or ITM.

- **Vega** measures how much an option prize increases as the underlying market volatility increases. Vega is positive for long calls and long puts, and negative for short calls and short puts. In fact, if we are buyers we are favored by the increase in volatility (positive vega) if we are sellers we are disadvantaged (negative vega).



(a) Credit Calendar Spread



(b) Debit Calendar Spread

Figure 2.8: Calendar option spreads (e.g. strike price 100 and 40 to 160 days to maturity)

- **Theta** measure the loss in value of the contract from one day to the next. It is maximum for ATM options and decreases to almost zero for OTM and ITM options. On days when the market is closed theta always acts in favour of the seller.

After this general analysis we can state the following reports.

In the money options has: an high delta practically speculating at the underlying with a tendency to increase as the deadline approaches; a low range and

decreasing as the deadline approaches; a low theta and falling as the deadline approaches and a low vega and decreasing as the deadline approaches.

At the money options has: an average delta which tends to remain constant even with the passage of time; an high range and rising sharply as the deadline approaches; an high theta and falling as the deadline approaches and an high vega and decreasing as the deadline approaches.

Out of the money options has: a low delta and falling as the deadline approaches; a low range and decreasing as the deadline approaches; a low theta and falling as the deadline approaches and a low vega and decreasing as the deadline approaches.

2.6 Platforms

A trading platform is basically a software tool through which it is possible to make financial investments and monitor accounts through financial intermediaries. Now a days there are a wide number of platforms, covering almost all types of markets.

The most important features are to enables investors and traders to place trades, monitor accounts and real-time quotes with charting tools and even premium research. Oftentimes, various trading platforms offer their members the possibility of using a demo account, that is basically an account for making virtual investments. In this way, there is no risk of losing money and it is very helpfully for beginners.

2.6.1 Competitors

For our purposes it is necessary to analyze some competitors that allow us to operate within the derivatives market. In particular, among the most important software are: FiutoBeta, OptionRuler and OptionVue. In this section we will analyze these platforms describing their main characteristics, costs and relative pros and cons.

FiutoBeta

FiutoBeta is a trading platform developed by PlayOptions Srl, an Italian company operating in the trading sector. The platform is made up of different sections dedicated to different functions, all reachable through the vertical bar on the left. In particular, the most important sections are:

- **Underlying** section, where is possible to load the underlying on which you want to operate and view some general information such as price history, open interest and volumes.
- **Options** section, where there are all the option contracts, call and put, for all the maturities and all the strikes available on the selected underlying. It

Figure 2.9: Underlying section on FiutoBeta

is therefore possible to add new positions to the current strategy and also viewing the payoff in real time.

- **Analysis** section, where you can analyze the current strategy and evaluate the probability of success
- Search and selection: where you can search for predefined strategies (e.g. bull call spread, butterfly, etc ...) and apply them to the selected underlying.
- **Tools** section, which includes all the useful tools to perform calculations of percentage variation, greeks and implied volatility in real time and insert alerts to monitor price trends.
- **Portfolio** section, where you can monitor the global trend of your capital by viewing the performance of all currently active strategies.

FiutoBeta is suitable for starting a new learning path in financial derivative instruments such as options, but also to provide support for the first options strategies, from the choice of the underlying to the control of the portfolio.

Pros

- Free software and data feeds
- Possibility to export your strategies

Cons

- Limited number of option contract per maturity date

OptionRuler

OptionRuler is an italian trading platform where is possible to carry out simulations and set up the purchase and sale strategies of options that are more consistent with price expectations.

Figure 2.10: Grid view in OptionRuler

The main screen is called Grid and is divided vertically into three sectors: call, strike price and put options data. Each row of the grid is referred to a single strike price with two option contract on for the call, on the left, and one for the put, on the right. Is possible to open the order form by clicking on one of the two sides of the option row and after that is possible to buy or sale the selected option and add a new position to the current strategy. After the Grid view there are several other screens, such as:

- the Calendar view that allows users to compare the median prices of call and put options of different maturities.

- the Strategy view that allows users to evaluate the profitability of the current strategy.
- the Portfolio view that allows users to manage temporary or opened orders

The platform does not charge account maintenance fees but commissions for European market are around € 5 per executed order and for non-European amounted to 9$. It is a service that embraces a very broad target of investors, ranging from beginners to those who are more experienced in the sector.

Pros

- Commission structure suitable for all types of investors.
- Possibility to connect the account to various external software.
- Great and fast customer service available 6 days a week, 24/24.

Cons

- High commissions on foreign markets.
- Most of the options are not available.
- Limited offer of foreign products and markets.

OptionVue

OptionVue is an advanced options trading software that includes some important features such as option matrix, highly customizable charts, reporting and modeling but also tools for finding the best trade strategy.

More in detail, the options matrix (Figure 2.11) is useful to look per each option contract, for a particular asset, all the pricing data, premium, volatility and some relevant greeks. There is the possibility to search for multiple dates and strike prices and open new positions for the current option strategy. From these positions OptionVue will calculate profit and loss that can be displayed by some advanced charts and permit comparison with other option strategies. Furthermore, there is a TradeFinder tool for scanning different options strategies and finding the best one that match some initial condition and filters.

Generally, is more suitable for experienced options traders and from the point of view of costs is a very expensive platform, with the basic fee that starts from 1000$ per year without the cost of data feeds and accessory modules.

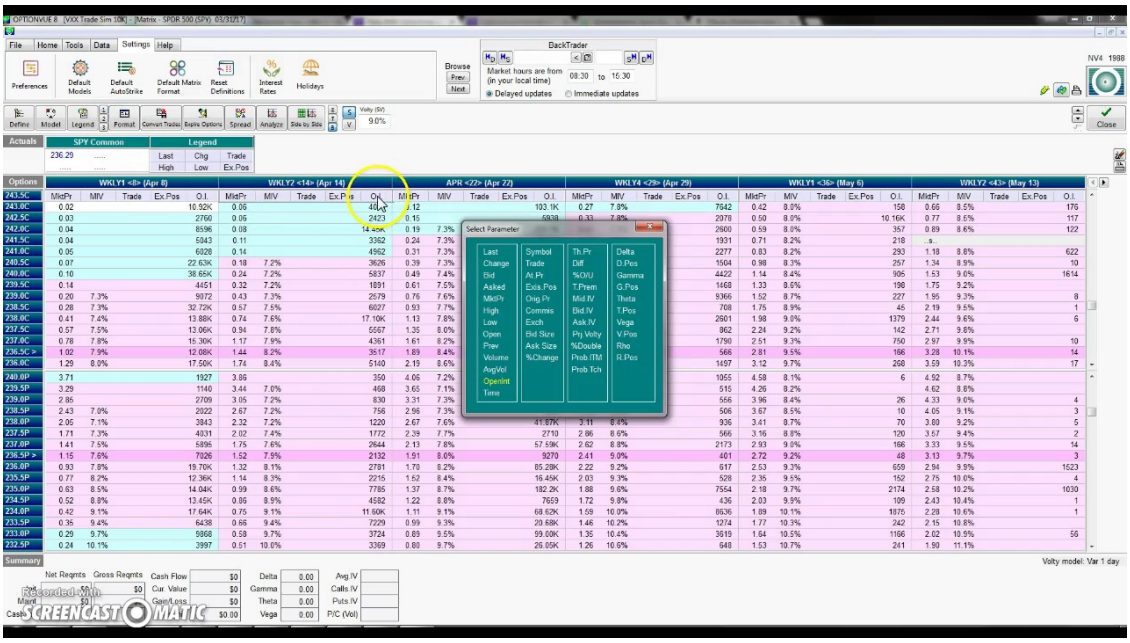


Figure 2.11: Option matrix feature in OptionVue

Pros

- Model implied volatility for any asset
- Detailed, interactive profit and loss charts
- Access to model parameters, including volatility calculations
- Includes trade tracking and reporting tools
- Interface and options matrix are fully customizable

Cons

- Very expensive and does not include data feeds

Chapter 3

Essentials

The idea is to develop a trading platform that includes the main features offered by competitors analyzed previously. In particular, it will be necessary to provide a section that ensures the operation on the main derivatives markets and assets with the consequent management of the strategies created by the user and the corresponding virtual portfolio.

In this chapter we will explain in more detail these requirements, the main data sources for the assets we are interested in, the technologies chosen for the implementation and the final architecture of the platform.

3.1 Requirements

Requirements is about defining the product properties before starting development and are divided in functional and non-functional requirements and should be both complete and consistent. They should include descriptions of all features required and should be no conflicts or contradictions in the descriptions of the system features. In reality, producing these documents is not at all trivial and many times is possible to fall into errors such as inserting features described in an incomplete or ambiguous way, entering redundant information or omitting important details. For this reason, predefined software engineering techniques have been developed to better formalize the requirements and reduce the occurrence of these errors. Some of these techniques will be presented below to identify the actors involved and the characteristics of functional and non-functional requirements.

3.1.1 Stakeholders

A stakeholder is a role or person that has an interest in the system to built and may affect or get affected by the outcome of the project. For example, a stakeholder can be the user or several user profiles that use the system and the supplier that pays for it but also the administrator, business analyst and developers. Listing the

relevant stakeholders is essential to consider relevant points of view, and therefore relevant requirements, for a system. Looking at Table 3.1 is possible to notice that many stakeholders are involved in the process

Name	Description
Developers	Do not use the application directly. They are involved in the development process of the app.
Users	Person using the app directly. After registration can search, view markets and operate with a virtual portfolio
Admin	Manages profiles and markets. Can decide to enable or disable some markets inside of the application
CBOE System	Provide REST API for downloading information about markets with options and futures contracts.
CME System	Provide REST API for downloading information about markets with options and futures contracts.
EUREX System	Provide HTML pages for downloading information about markets with options and futures contracts.

Table 3.1: Stakeholders

3.1.2 Functional and Non-functional

To define these requirements, a software quality model defined by the ISO[1] must be followed. In particular, the ISO9126/25010[9] standard defines six properties of software systems: functionality, reliability, usability, efficiency, maintainability, portability.

Functional requirements, includes only the first property and should provide a description of services that we are going to implement and should contains terms of the application domain, explain the behaviors provided by the system and must be understandable to the customer. The difficult part is to distinguish one requirement from another and for this reason is important to assign for each functionality an ID in order to easy identify it and trace throughout the life cycle of the project (Table 3.2).

Non-functional requirements, includes the remaining five properties and for this reason may be more critical than functional requirements. They are not directly connected with the functionalities to be implemented, but refer to operating methods and constraints, such as response times, supported platforms, choice of languages, required resources, tools and various implementation techniques and

[1]International Organization for Standardization

must be measurable. As happens for the functional requirements they are identified with a unique code and in addition to the description, it is necessary to specify their type associated to the ISO properties and which functional requirements they refer to (Table 3.3).

ID	Description
FR1	Sign-up users by email and password
FR2	Login and logout users and admin users by email and password
FR3	View the Markets page with a list of all the markets available
FR3.1	Search markets form the list with input text filter
FR4	Select a market from the list
FR4.1	Browse information about prices, options and futures contracts
FR4.2	View chart for historical prices, volumes and open interest of the underlying
FR4.3	View chart for statistics about volatility
FR4.5	View chart for statistics about open interests of options contracts
FR5	Create an empty strategy with a name and selecting the target market
FR6	Delete an existing strategy
FR7	Update the name of an existing strategy
FR8	Select the strategy you want to operate on
FR8.1	Add or remove temporary positions on options or futures contracts
FR8.2	Open positions from temporary positions on options or futures contracts
FR8.3	Close positions from open positions on options or futures contracts
FR8.4	View the strategy costs and profit
FR8.5	View chart for the "Payoff"
FR8.6	View chart for the "Greeks"
FR8.7	Enable or disable the "What-if" mode
FR9	View the Strategy page with a list of all active strategies
FR9.1	Search strategies form the list with input text filter
FR9.2	Select a strategy from the list for having a short summary of it
FR10	View the Portfolio page and balance information
FR10.1	View chart for performance history
FR10.2	View table of all closed strategies
FR10.3	Active or Deactive closed strategies

Table 3.2: Functional requirements for the platform

ID	Type	Description	Refers to
NFR1	Usabilty	Application should be used with no training by any user	All FR
NFR2	Performance	All functions should complete in < 0.5 sec	All FR
NFR3	Portabilty	System must work on Chrome from version 70, Firefox from version 65, Safari from version 10, Edge from version 72	All FR
NFR4	Portabilty	No installation is needed	All FR
NFR5	Portabilty	The support for geolocalization is not required	All FR
NFR6	Security	Accordingly to the GDPR law (art 13 and 14), since data won't be used for commercial goals and won't be sent to third parties, an informative must be provided to users and gas stations' owners, but him/her consent is not needed	FR1
NFR7	Localisation	Decimal numbers use . (dot) as decimal separator	All FR
NFR8	Localisation	Currency are € (Euro) and $ (Dollar)	FR4.*, FR8.*, FR10.*
NFR9	Reliabilty	Downtime allowed is of one hour per year	All FR

Table 3.3: Non functional requirements for the platform

3.2 Data sources

The data source is the starting point for each application and finding the right source is one of the most important point, but before that it is also essential to make some consideration of which type of data sources is the best to work with.

3.2.1 Types of data sources

In general data comes from a diverse type of data sources. The most common of these are:

- **Databases** an organized collection of data, generally stored and accessed electronically from a computer system. Where databases are more complex they are often developed using formal design and modeling techniques.

- **Flat files** a plain text file, or a binary file where records follow a uniform format, and there are no structures for indexing or recognizing relationships between records.

- **Web services (WS)** a service running on a computer device, listening for requests at a particular port over a network, serving web documents (HTML, JSON, XML, images), and creating web applications services, which serve in solving specific domain problems over the Web (WWW, Internet, HTTP)

- Other sources such as RSS feeds

For our purpose, web services are the easiest type of data source to find and use. In particular, in recent years efficient guidelines for WS called REST API[2] have spread, which makes accessing the web resources exposed by the service even more intuitive. For this reason our research has focused on finding valid APIs that can track all information about finance summary, stocks, quotes, options.

3.2.2 Markets web services

After the analysis of different types of data sources, we are now looking for the web services exposed by the main exchanges presented in the section 2.1 that are CBOE, CME and EUREX.

[2]an application programming interface (API) that queries data, parses responses, and sends instructions between one software platform and another

CBOE System

From its official website CBOE exposes various services that allow the user to view a lot of information about US Stocks and Indexes Options and many other market information in delayed time (10-15 minutes). In particular, there are two reference public REST API:

- **GET /symbol_book/option-roots.json** to download all options catalogue info. The response is an array of JSON[3] objects and each of them includes the identifier (option-root) for downloading further information through other web services.
- **GET /options/{option-root}.json** to download updated information about a single option-root. In particular, the response contains information like open, close prices of the underlying and an array of options contract with their relative info (e.g. put or call, strike price and maturity date) and prices (Listing 3.1).

The first API is not used by the application, but is useful to have a list of how many and which markets CBOE includes. The second one, instead, is used to download all information about a single market. Indeed, with one call is possible to obtain all prices and options prices for all expiration of the input market and this operation is done every 15 minute without any rate limitations from CBOE System.

```
GET /options/AAPL.json
{
   "timestamp":"2021-03-14 13:19:37",
   "symbol":"AAPL",
   "data":{
      "high":121.1699,
      "close":121.03,
      "open":120.31,
      "low":119.16,
      "volume":40992675,
      "last_trade_time":"2021-03-12T15:59:59",
      ...
      "options":[
         {
            "high":0.0,
            "prev_day_close":56.9750003814697,
            "open":0.0,
            "open_interest":13.0,
            "low":0.0,
            "option":"AAPL210312C00065000",
            "volume":0.0,
            "last_trade_price":56.93,
            "last_trade_time":"2021-03-11T10:47:59",
            ...
         },
```

[3]JavaScript Object Notation

```
            ...
        ]
    }
}
```

Listing 3.1: Example of option-root response for AAPL stocks

CME System

It is very important for our goal to download information about options and futures contracts of the most important indices such as S&P 500, Nasdaq and Russell 2000. The main provider of this information is CME, which from its website displays this content publicly and free. In particular, among all the APIs that CME presents, it is important to describe the following:

- **GET /services/product-slate** to download all the list of products or underlying that CME provides to the public. Each element of the list contains descriptive information of the product and its identifier, called product_id, which will be essential to obtain the product related futures and options contracts.
- **GET /Quotes/Future/{product_id}/G** to download all the futures contracts related to a particular product for each available expiry date and their prices.
- **GET /Quotes/Option/{product_id}/G/{expiration_id}** to download all the options contracts related to a particular product and expiration date (Listing 3.2). In particular, the expiration_id is an encoding of the typical quarterly deadlines related to options and futures: March (H), June (M), September (U), December (Z).
- **GET /Volumes/Details/{product_type}/{product_id}/{last_trade_date}/P/** to download all statistics data about volumes and open interest of the input future or option contract for a past trading date. Indeed, the API require a product_type that could be "F" or "O" for future and option respectively and followed by its identifier and the date they were traded with "yyyymmdd" format.

The first API it will not be called automatically by the application but it is important to download the CME product catalog and initialize our internal catalog with the selected markets. The remaining APIs will instead be contacted at regular intervals of 15 minutes during the trading hours for updating market, futures and options data.

```
GET /Quotes/Option/133/G/H1
{
   "tradeDate":"18 Mar 2021",
   "optionContractQuotes":[
      {
         "strikePrice":"391000.0",
         "strikeRank":10,
         "put":{
            "last":"14.00",
            "priorSettle":"3.35",
            "open":"2.00",
            "close":"-",
            "high":"15.50",
            "low":"1.00",
            "volume":"3,771",
            ....
         },
         "call":{
            "last":"18.25",
            "priorSettle":"67.25",
            "open":"77.00",
            "close":"-",
            "high":"77.00",
            "low":"18.00",
            "volume":"363",
            ...
         },
         "underlyingFutureContract":"ESH1"
      },
    ...
   ],
}
```

Listing 3.2: Example of option quotes response for future E-mini S&P 500

EUREX System

Unfortunately, the public APIs only expose reference information regarding products, contracts and other instruments for trading but do not go into detail on options and futures. The only free way to access this information is through their dedicated HTML pages by using some web scraping[4] techniques.

In this way, is possible to download various information about options and futures contracts of the most important European indices, such as EuroStoxx50 and DAX.

3.3 Architecture

In this section will be briefly described the architectures chosen to better understand the organization of the technologies used and how they interact with each other for

[4]is a computer technique of extracting data from a website by means of software programs

the realization of the trading platform.

3.3.1 Client and Server

The client-server architecture (Figure 3.1) has been chosen as it is the most recommended and widespread architecture that allows users to interact with a web application through an internet connection and in particular with a web browser. More in general, it is an architecture where a computer, **client**, connects to another computer, **server**, to use one or more exposed services. The server, therefore, is a component of processing and management of information traffic and provides any type of service to other components, clients, who request it through specific network protocols (e.g. HTTP). In addition, for the services exposed by the server there are different software architecture and guidelines for their implementation. In recent years, however, there has been a strong trend in the use of Representational State Transfer API (REST API) guidelines.

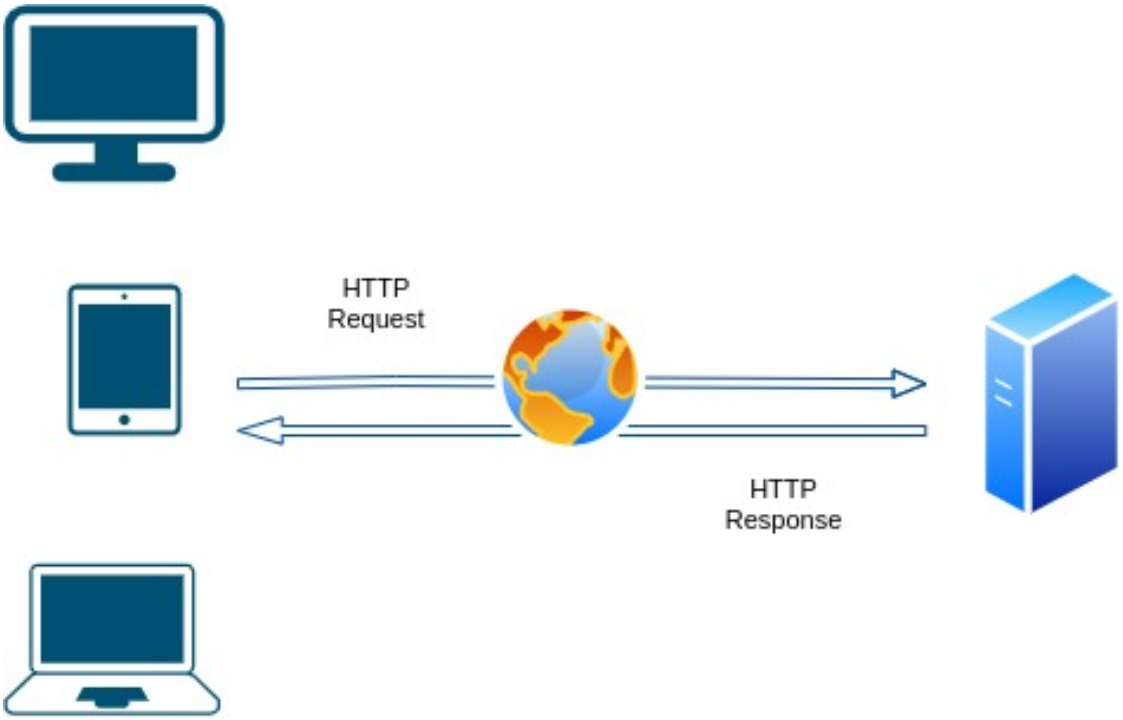


Figure 3.1: Client-server architecture

Rest API

Its operation involves a well-defined URL[5] structure that uniquely identifies a resource or set of resources and the use of specific HTTP methods for getting (GET), adding or editing (POST, PUT, PATCH, DELETE) information and other purposes (OPTIONS, etc.).

Table 3.4: Relation between the URL and HTTP Methods in REST API architecture

URL	HTTP Methods			
	GET	POST	PUT	DELETE
Collection (e.g. http:// api.com/resurces)	Return a resource list with some details on items belonging to the collection	Create a new item in the collection	Replace the entire collection with another collection	Delete the entire collection
Element (e.g. http:// api.com/resources/1)	Return information about the item identified with the code "1"	-	Replace some information of the item identified with the code "1"	Delete the item identified with the code "1"

3.3.2 Server-side containerization

The server-side component of the web application can be developed and deployed using Docker containers. This architecture enable you to package an application and all its dependencies without having to worry about the hardware or software features of the physical server. That is because a container carries all its dependencies with it, wherever it goes. Furthermore, with containers it is possible to run multiple web applications on the same physical server in a isolated way and significantly reduce data center management cost and administrative overhead as IT organizations.

For this reason, we decide to develop the trading platform by using Docker containers. Looking at Figure 3.2 is possible to see all required application organized and connected with each other by the Docker compose tool.

[5]Uniformed Resource Locator

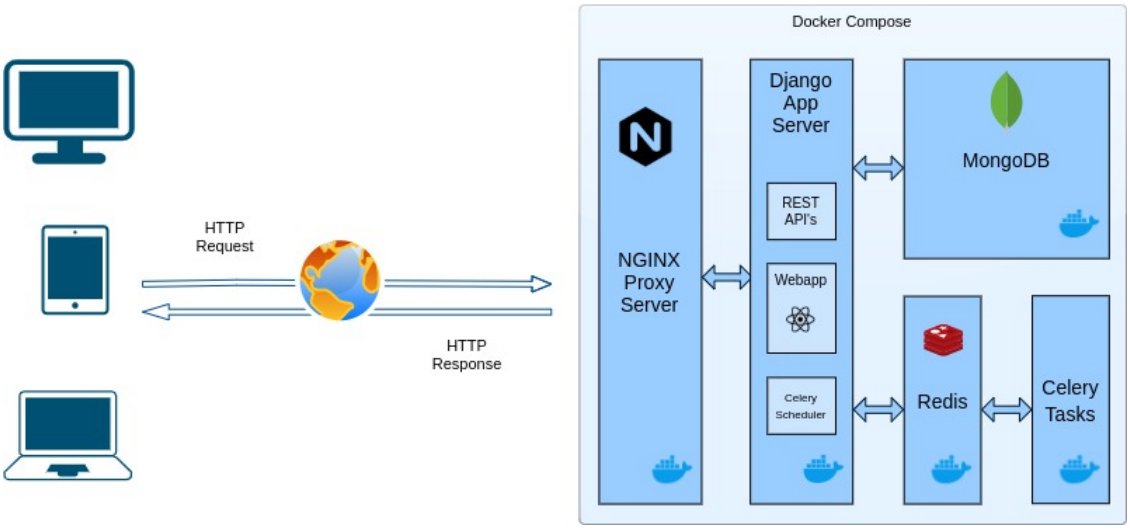


Figure 3.2: Docker compose architecture

3.4 Technologies

Starting from the architecture described, the technologies used can be separated into three distinct parts: Django framework for server-side development, React framework for client-side development and MongoDB for database management. In this chapter these technologies will be described showing their main features and extensions with in addition a small analysis on their alternatives.

3.4.1 Django

Django was born in 2003 as an open source web development framework [6] based on Python. It offers standard methods for developing new web applications in a rapid and effective way and allows developers to include inside a project some shared core applications for increasing modularity and reducing the development time. Django is widespread among big companies like Instagram, Pinterest and BitBucket and used by tens of thousands users with a plenty of educational contents available.

Python

Python is a popular high-level dynamic programming language with a lot of features. In particular it provides an intuitive object orientation, strong introspection capabilities, readable and very clean syntax. It has a very high level dynamic data types, supporting hierarchical packages and full modularity. In addition, there is a

[6] code library for building scalable web app or web services

huge community behind Python that develop a lot of extensive standard libraries and third party modules for every possible task also written in different languages like C, C++, Java with Jython or .NET languages for IronPython.

For these reasons Python is also used in variety of application domains such as Web and Internet development, Desktop GUIs, Education and more and it is more attractive than programming languages such as Java, Ruby, Perl and others. Finally, Python is easy to learn thanks of the well written documentation and it is freely distributed and usable even for commercial use since it is under open source license.

Architectures

Django helps developers to follow best practices and maintains a clean project. Each application should be independent with others and developed in one place without repeating it code in other applications.

Furthermore, Django is mainly based on Model-View-Template (Figure 3.3) software design pattern for developing a web application:

- the **model** is responsible for maintaining the logical data of the application and acts as a mediator between the website interface and the database
- the **template** is a static file that describe with a special syntax how the content of one or more models will be inserted
- the **view** handle the user interaction and interact both with the model and the template in order to give a response to the user

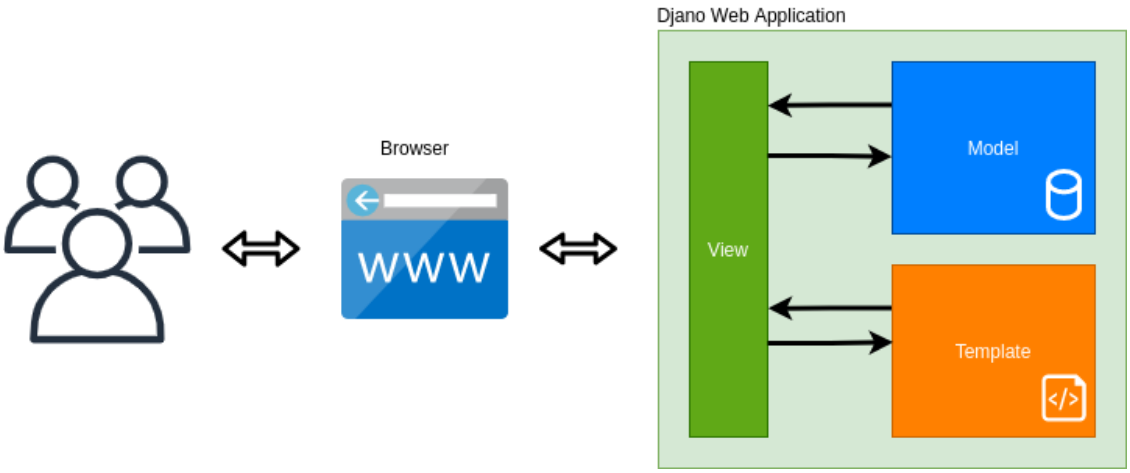


Figure 3.3: Django MVT Pattern

Pros and cons

There are many **features** that can be found in a newly created Django project and since from the beginning you can directly work with an highly customizable admin interface that allows CRUD [7] operations with the database.

Furthermore, there is a built-in authentication system that handles user accounts, groups and cookie-based user sessions and includes security techniques to avoid attacks like CSRF, Cross-site scripting, SQL injection, and Clickjacking. Provides support for multiple cache mechanisms, end-to-end application testing and translating text into various languages, local formatting dates, times, numbers and timezones. Also, multiple external libraries can help, like REST Framework that provide supports for building APIs with validators and authentication protocols with few lines of code.

On the other hand, there are also some relevant **drawback**, since Django handle a single API request per time. Furthermore, Django Templates failed silently therefore it is more difficult and time consuming to debug, routing requires some knowledge of RegExp[8] and in general there is high learning curve to master it.

Alternatives

There are several frameworks written in Python that can be evaluated as an alternative to Django but few have its community strength and diffusion in enterprise products. In particular, we will analyze the main features of the following quite popular alternatives.

Web2py is a framework easy to use which focus on security and development speed. It cover all functionalities that Django has since it offers a lot of features out of the box such as a web server, admin panel, database, grid widgets or wiki and both can be used to fulfil the same tasks. The only difference is that Web2Py might be a little harder to study and to find help in case of trouble since it is younger and has a smaller community than Django.

Flask it has a simple but extensible core framework, also called micro framework and has no dependencies on external libraries. This framework offering basic features of web app but more important are the huge quantity of extensions that allows to add functionalities for form validation, object-relational mappers, various authentication technologies, file upload management and more other tools. However, the choice of extensions and their insertion already requires more work and

[7] Create, Read, Update and Delete operations

[8] Regular Expression

therefore increases the probability of making mistakes. So this framework is recommended when a high level of granularity is desired or you need to build small application in a short time.

Pyramid is a lighter version of Django. It includes features like authentications and routing while for other requirements such as database management or templating it is necessary to install external plugins. It is convenient to use when dealing with large systems that require flexibility and lightness but on the other hand it requires more effort during the development process.

3.4.2 React

React is a JavaScript library developed by Facebook in 2013 primarily used for developing user interfaces. This flexible front-end solution does not enforce any particular architectural pattern for handling data, it focuses only on the creation of views by a composition of React components. These are mostly written in JSX (JavaScript XML) syntax that allows developers to write elements containing HTML and JavaScript at the same time and are the smallest building blocks within an inner state and props that make them flexible and reusable. Data flows downwards through the component tree using props and also callback functions to interact among them. In large React applications this composition becomes deep, tightly coupled, less maintainable and can lead to "props-drilling". This is one reason why architectural patterns, like Redux pattern, are necessary for complex React applications.

JavaScript

JavaScript was originally developed by Netscape Communication Corporation in 1995 for the clien-side web application development. It is a type safe and dynamic scripting language and the code is not compiled, but executed by the interpreter contained in the user's browser. The syntax is similar to that of the languages like C, C++ and Java and also provide multi-paradigm language (e.g. object-oriented, imperative and functional programming styles).

JavaScript has a standard called ECMAScript and modern internet browsers are updated and fully support the ECMAScript v11 (2020).

React Redux

Redux is a JavaScript library used for implementing a pattern similar to the MVC pattern with the purpose to have more control of the data flow of the application (Figure 3.4). Redux uses a single **store** that acts as a single source of truth for the whole application. Views subscribe to the store and re-renders when it changes,

which makes the data flow uni-directional. Furthermore Redux has the concept of **reducer** as a pure functions that take the previous state from the store as an argument and an **action**, an object that includes a set of parameters, and returns the new state of the store. Reducers functions have a set of guidelines:

- They do not rely on any other data than the data sent in as parameters
- The arguments to a pure function are seen as immutable and should not be changed

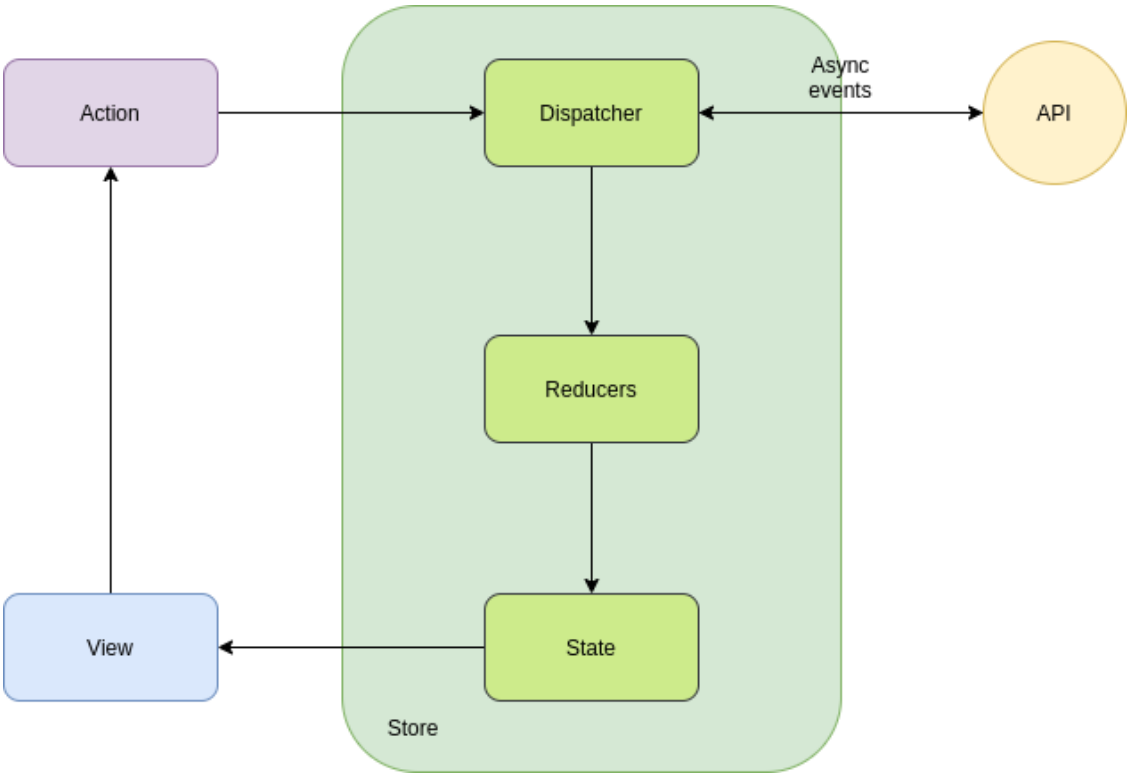


Figure 3.4: Redux architectural pattern

React Stock-charts

React Stock-charts[12] is an open-source library built with React and D3JS libraries. Provides features and flexibility to create stock charts that compete with the likes of the ones provided by commercial trading systems. Indeed, this library provide predefined Area, Bar, Line and Candelstick charts but it is also possible to add custom chart components and indicators or access the svg elements and styling with CSS. Furthermore, provides fast performance to pan and zoom actions also for mobile device.

Alternatives

Below, we will briefly describe some alternatives to using React as a framework for the front end. It is important to underline that among all the alternatives React is the best choice if you want to create a stable and well-structured application in a short time for both simple and complex projects thanks to the use of React Redux.

Angular is a well furnished framework for web application development based on TypeScript and mainly used for building single-page web applications (SPAs). It is component-based and structured in Modules, Components and Services and also include HTML template syntax with special directives to output reactive data and may render multiple elements. Angular is best suitable for large-scale enterprise applications and advanced projects that needs complex infrastructure.

Vue is a high performance, front-end Model-View-View-Model open-source JS library. Its optimal user experience and versatility on a web application have made it one of the most popular frameworks. Thanks to its quick learning curve is suitable for solving short-term problems but also for building web apps with animations, interactive elements or prototyping.

3.4.3 MongoDB

MongoDB[10] is a NoSQL DBMS[9] developed in C++ by 10 Gen in 2007 and become an open-source project in 2009. Thanks to the absence of join, queries are simpler and faster with high performance in reading and writing. Furthermore, the most consistent readings can be distributed in multiple replicated servers. The most important features are:

- document-oriented, that is, data are stored in the form of JSON-style documents with dynamic schema, according to a very simple and powerful structure that also allows the representation of complex hierarchical relationships through nested documents and arrays
- allows indexing of any attribute
- easily allows data replication through high scalable network
- excellent horizontal scalability without compromising any functionality

[9]Non-relational Database Management System

Pymongo

PyMongo[11] is a Python distribution containing tools and drivers for working with MongoDB, and is the recommended way to work with MongoDB from Python.

NoSQL vs SQL

NoSQL or non-relational DBMS are document-oriented databases, where the structure of the data to be stored is not defined and the types of data may vary from document to document. This makes these databases very flexible and suitable for projects where it is difficult to define the real data structures but complicates the job when it is necessary to exploit the relationships between documents. On the contrary, SQL or relational databases save data in structured and well-defined tables, rejecting in input anything that does not respect their rules, therefore ideal in case it is necessary to model many relationships between data because they are well known.

Alternatives

In this section we will discuss some relational and non-relational databases as alternatives to MongoDB.

PostgreSQL it is a popular, free SQL database with 30 years of continuous development with good improvements in terms of scalability, relaiability and availability. It is used by a large number of developers and can be run in very different platform also in cloud. It is ACID [10] compliant and highly suitable for financial applications, therefore ideal for online transactions workloads.

MySQL is a popular community-driven relational DBMS system able to run on many distribution of UNIX and Linux. It makes database administration easier and more flexible and can handle any amount of data, up to as much as 50 million rows or more. Offers built-in tools for query analysis and space analysis and support memory storage engine for frequently used tables. A great choice for structured data with the priority for high data security.

DynamoDB is a NoSQL database developed by Amazon Web Services (AWS). Its supports auto sharding and load-balancing and it is ideal for applications that stores a large amount of data with strict latency requirements but it is mandatory to use it on the AWS platform since it is one of their exclusive product.

[10] Atomicity, Consistency, Isolation, and Durability properties that any transactions must have.

Cassandra is a highly scalable open-source NoSQL database of the Apache software foundation. It is suitable for massively scalable systems and real-time analytics applications with a large amount of data to process. But it does not allow ad-hoc queries and it has limited support for aggregation ones.

In front of these valid alternatives, MongoDB was chosen because today it is the best DBMS when looking for high flexibility and adaptability to real business world situations and requirements. Furthermore, it is a very easy to scale up and down, high availability with replica sets and high fault tolerance.

3.4.4 Celery

Celery[1] is a simple, flexible and reliable asynchronous task queue[11]. Celery supports both real-time processing and task scheduling through the Celery-beat tools. Furthermore, supports concurrent task executed either asynchronously or synchronously. Although Celery is written in Python, the protocol can be implemented in any language and can be easily integrated with a vast number of web frameworks (e.g. Django).

Redis

Since Celery communicates via messages it usually uses a separate service called a message broker to send and receive messages between application and workers. For this purpose, Redis[13] is one of the most recommended message brokers for its integration with Celery.

3.4.5 Docker

Docker[4] is an open source containerization platform invented by Solomon Hykes in March 2013. Docker enables developers to package applications into a Docker **image**: a lightweight, standalone, executable package of software that combine application source code with all the operating system libraries and dependencies required to run the code in any environment. A Docker image become **container** when they run on Docker Engine[12].

Docker uses a client-server architecture for handling request from the user and managing images and containers (Figure 3.5). In particular, on the server side a Docker "daemon" is active which has the task of receiving and executing all the requests of the Docker client through the CLI[13] if both are on the same host or

[11] task queue is a mechanism used to distribute work across threads or machines

[12] Open source containerization technology for building and containerizing your applications

[13] Command Line Interface

through the REST API if the server is remote. The Docker registry, on the other hand, is a platform that everyone can use to upload or download their public or private images.

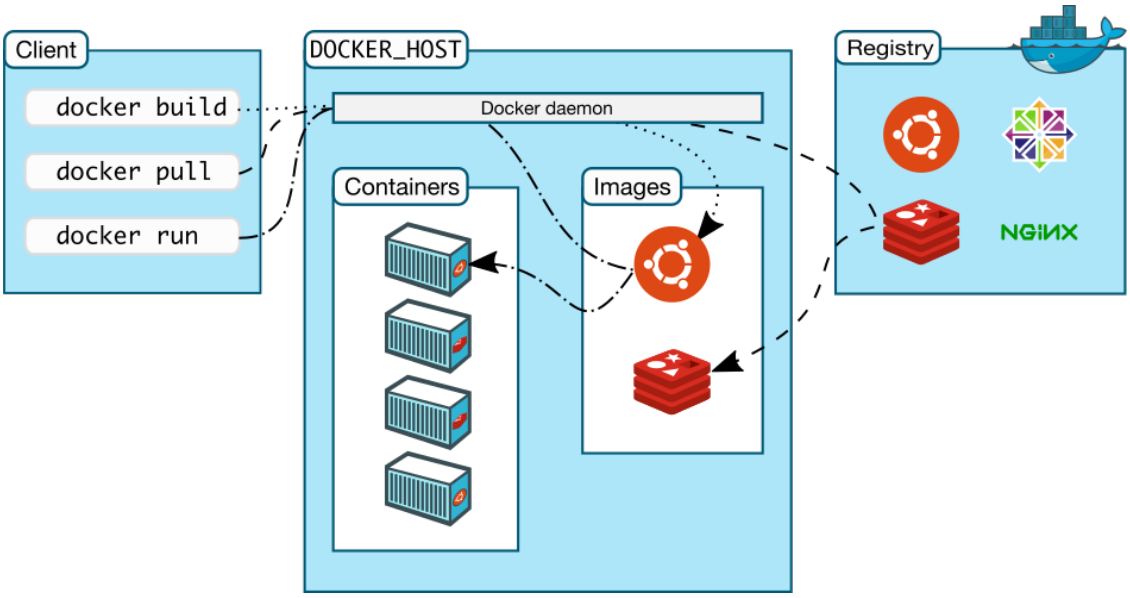


Figure 3.5: Docker client-server architecture available on this **link**.

Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications, also called services. These can be configured through a YAML[14] file per each environment (e.g. production, system or develop) and then, with a single command, is possible to create and start all of them. More about Docker Compose provides these features:

- Multiple isolated environments on a single host
- Preserve volume data when containers are created
- Only recreate containers that have changed
- Variables and moving a composition between environments

[14]Data-orientated human readable serialization language

3.4.6 Nginx

Nginx is an HTTP web server that can also be used as reverse proxy, mail server, load balancer and generic TCP/UDP server. Nginx uses an architecture event driven requests management that allows the server to react to hardware signals generated to the operations of input and output. This makes it extremely efficient in managing large quantities of competing requests.

Chapter 4

Implementation

Here, the implementation of the trading platform will be dealt with in more detail. In particular, the data models used, the asynchronous tasks for downloading financial information and the services exposed for the client will be described.

4.1 Database

As already specified in the previous section 3.4.3, for data persistence it is used a non-relational database and specifically MongoDB. Therefore, in this section we will describe collections and documents used for saving user and markets information.

4.1.1 Collections

From the beginning we chose to not define Python classes but to use JSON dictionaries directly to compose each collections. In this way, one document can have a different structure from one other inside of the same collection and this flexibility reduce the duration of the developing process and require a low effort at the beginning to identify all data structures.

User

Collection of user documents that contains all the information that identifies and describes each single user such as email, name and surname but also information regarding their operation within the platform as we can see in Table 4.1.

Market

Collection of market documents that represents general markets information and they relationships with options and futures (Table 4.2). In particular, the list of

Table 4.1: User document description

Fields	Description	Type
id	Unique identifier of the user inside of the database	Number
email/username	Unique identifier for the authentication	String
password	User associated password for the authentication	String
first_name	Name of the user	String
last_name	Surname of the user	String
is_superuser	If the user is authorized to be a super user inside of the application	Boolean
is_active	If user is still active to operate with the application	Boolean
last_login	Last time the user logged in the application	Datetime
date_joined	Date and time the user joined in the application	Datetime

expiration that are inside of each market and described in Table 4.3 indicates all options and futures that are available for trading.

Chain

Collection of chain documents that contains all options that refer to the same market and the same time to maturity (Table 4.4). Therefore, a chain is identified as a composition of the market symbol (e.g. SPX), expiration symbol (e.g. EOM, End of Month) and expiration date in *dd-mm-yyyy* format (e.g. 08-20-2021). Each element in the options list contains, for the same strike, the information of both call and put options (Table 4.5 and 4.6). Globally, a single option is identified by the **contract** field composed by the identifier of the chain as explained above (e.g. SPX-EOM-20-08-2021), the contract type (e.g. PUT or CALL) and the left zero-padded strike price, with a fixed length of 10 characters (e.g. 0000100000)

Future

Collection of future documents that contains information on a future referring to the same market and the same time maturity (Table 4.7). Therefore, a future is identified by the **contract** field composed with the market symbol (e.g. SPX), expiration symbol (e.g. EOM, End of Month) and expiration date in dd-mm-yyyy format (e.g. 08-20-2021). Furthermore, to keep the same format used for option

Table 4.2: Market document description

Fields	**Description**	**Type**
id	Identifier for outsourced systems like CBOE, CME and EUREX	String
groupId	Identifier for grouping the market with other related markets (relationship with Group data model)	String
symbol	Symbol of the market used inside of the application	String
label	Name of the market	String
exchange	Symbol of the exchange to which the market belongs	String
country	Symbol of the country to which the market belongs	String
currency	Predefined currency of the market	String
template	Type of the market (index, stock, index futures...)	String
exposition	Exposition value for related financial options	Number
dividendYield	Dividend yield value of the market	Number
expirations	Array of expiration data models	Array
underlying	Information related to the market's underlying	Object

Table 4.3: Market expiration document description

Fields	**Description**	**Type**
symbol	Symbol of the expiration	String
label	Label of the expiration	String
dates	Array of Datetime data types	Array

contracts the system append at the end the contract type equal to FUTURE and a strike price equal to 0.

Group

Collection of group documents that used to maintain relationships between different markets. In fact, a group of markets can be composed for example by S&P 500, E-mini S&P 500 and Micro emini S&P 500, as all three refer to the same S&P 500 index. Modeling these relationships is important for the creation of strategies that allow the user to operate within the same group of markets (Table 4.8).

Table 4.4: Chain document description

Fields	Description	Type
exchange	Symbol of the exchange	String
symbol	Symbol of the market	String
expiration	Symbol of the expiration	String
date	Date and time of the expiration	Datetime
options	Array of option data model	Array

Table 4.5: Strike document description

Fields	Description	Type
strike	Value of the strike price	Number
put	Option object of type "put"	Object
call	Option object of type "call"	Object

Strategy

Collection of strategy document that are able to store all the operational choices that a user makes with the application for a specific group of markets (Table 4.9). Therefore, a strategy has a strong relationship with the **user** and the **group of markets**, but is globally identified with an auto-generated id since it is possible for a user to create multiple strategies for the same group. Inside of the document there is a list of all market positions that a user decides to open and close and each position refers to a single option or future (Table 4.10). They are characterized by the amount of contracts that the user decided to buy or sell and can take on three different states: temporary, open and close. A temporary position has no effect on the user's portfolio, namely no opening costs or possible profit or loss amounts are charged. Its purpose is purely indicative and allows the trader to understand what impact it can have on the payoff of the selected strategy. From a temporary position the user can decide to open, disable or delete it.

An open position has a specific cost on the strategy and consequently on the portfolio and cannot be canceled or deactivated, the only action allowed is the closing one. An open position is identified by concatenating the following values:

- contract identifier (e.g. CBOE-SPX-EOM-20210820-CALL-0004405000)
- timestamp in which the position was opened (e.g. 1628070767071)

in this way it is possible to open various positions for the same contract in different times.

A strategy also has a "whatif" logic that allows the user to modify some parameters of one or more positions like initial or final price, volatility, days to expiration and many others to perform simulations on the strategy payoff.

Table 4.6: Option document description

Fields	Description	Type
price	Price of the option	Number
last	Last price of the most recent trade	Number
open	The first trade price at the open of the most recent trading day session	Number
close	The final trade price at the close of the most recent trading day session	Number
settle	The final settlement price calculated at the end of the previous trading day	Number
low	The lowest trade price during the trading day	Number
high	The highest trade price during the trading day	Number
volume	The total number of contracts traded during the trading day	Number
openInterest	Total open interests of the option during the trading day	Number
type	Type of the option	String
state	State of the option	String
contract	Contract of the option	String

Portfolio

Collection of portfolio documents that are a virtual account that keeps the total balance of each user. All portfolio starts from a default initial value of € 100,000 and based on the results obtained by the various strategies performed by the user the balance may increase or decrease (Table 4.11). Currently there is a one-to-one relationship with the user and his portfolio and for that reason it is identified by his username.

Market History

Collection of documents that are uniquely identified by the symbol of the reference market and aggregates in an internal list all the prices that the market assumes every day with a maximum depth of two years (Table 4.12).

Chain History

Collection of documents that are uniquely identified by the composition of the symbol of the market, expiration type and date (as for chains) and aggregates all

Table 4.7: Future document description

Fields	Description	Type
exchange	Symbol of the exchange	String
symbol	Symbol of the market	String
expiration	Symbol of the expiration	String
date	Date and time of the expiration	Datetime
price	Price of the future	Number
last	Last price of the most recent trade	Number
open	The first trade price at the open of the most recent trading day session	Number
close	The final trade price at the close of the most recent trading day session	Number
settle	The final settlement price calculated at the end of the previous trading day	Number
low	The lowest trade price during the trading day	Number
high	The highest trade price during the trading day	Number
volume	The total number of contracts traded during the trading day	Number
openInterest	Total open interests of the future during the trading day	Number
type	Type of the option	String
contract	Contract of the future	String

Table 4.8: Group document description

Fields	Description	Type
type	Type of the group of markets	String
symbol	Identifier of the group of markets	String
name	Name of the group of markets	String
currency	Currency symbol shared by all markets present in the group	String

the options in an internal list with the relative prices per each day with a maximum depth of one month (Table 4.13).

4.1.2 Data analysis

To save and keep the chain and market historical data at the end of the day we decide to implement the bucket aggregation model[5]. Which consists in having a

Table 4.9: Strategy document description

Fields	**Description**	**Type**
_id	Unique identifier auto-generated by MongoDB	ObjectId
userId	Identifier of the user that create the strategy	String
groupId	Symbol of the group of markets which the strategy belongs	String
name	Name of the strategy given by the user	String
positions	Array of position objects	Array
created	Date and time when the strategy was created	Datetime
disabled	If the strategy is disabled for the portfolio	Boolean
closed	If the strategy has no more open position	Boolean
whatif	Object that contains all information for the whatif simulation	Object

single document for each market or chain in which to group in an array of sub-documents all the information of the same market or chain sampled day by day. The size of this array determines the number of days we want to aggregate and is commonly called the bucket size. In this section will be exposed some analyzes carried out for underlying the effects of this aggregation technique with three different sizes:

- one document per day, this mean no grouping and every day a new document will be created
- one document per week, this mean a weekly grouping and every 7 days a new document will be created
- one document per month, this mean a monthly and every 28 days a new document will be created

Assuming that every day it is necessary to keep the history of 160 chain documents and that on average a document occupies 800 kB, if the number of elements aggregated in a single document is increased, within a few days we obtain:

- less documents within the same collection (Figure 4.1)
- less total space occupied by the collection, due to the reduction of duplicate fields (Table 4.14)

Table 4.10: Position document description

Fields	Description	Type
id	Unique identifier of the position	String
contract	Contract identifier of the chosen option or future	String
active	If the position is disabled or not	Boolean
status	State of the position	String
quantity	Number of contract bought (positive value) or sold (negative value)	Number
exchange	Exchange symbol of the market	String
symbol	Symbol of the market	String
expiration	Symbol of the expiration type	String
date	Date of the expiration	Datetime
type	Type of the contract	String
strike	Strike price in case of the option contract	Number
price	Current price of the contract	Number
whatif	Object that contains all information at position level for the whatif simulation	Object
startDate	Date and time when the user open the position	Number
startPrice	The price of the option or future when the user open the position	Datetime
endDate	Date and time when the user close the position	Datetime
endPrice	The price of the option or future when the user close the position	Number

- more space in average occupied by a single document (Table 4.15)

In conclusion, it is important to find the right trad-off between the number of documents present within the same collection and the space occupied by a single document to avoid slowing down the execution of queries, reducing the number of duplicate information and not to exceed the space limits for a single document provided by the various databases (e.g. maximum 16MB for a single MongoDB document)

4.2 Async tasks

Asynchronous tasks are essential for maintaining and retrieving information on financial markets. In particular, for each chosen exchange it is necessary to have an

Table 4.11: Portfolio document description

Fields	Description	Type
_id	Unique identifier auto-generated by MongoDB	ObjectId
userId	Username of the user which the portoflio belongs	String
name	Name of the portoflio	String
value	Total balance of portfolio	Number
currency	Currency symbol used for the portfolio	String
created	Date of creation	Date and time
strategies	Array of strategies ids associated to the portfolio	Array

Table 4.12: Market history document description

Fields	Description	Type
exchange	Exchange symbol to which the market belongs	String
symbol	Symbol of the market	String
days	Array of day object which contains prices, volume and open interest per each day	Array

asynchronous task that retrieves the daily information of all the markets belonging to that exchange, with regular intervals of 15 minutes (Table 4.16). At each execution, all prices, volumes, open interest per market, future and chain of options are downloaded and processed in such a way as to have a certain uniformity between documents and after that the database will be updated. While for the maintenance of historical data, only one task performed at the end of the day is used, which aggregates the latest updated values in unique documents per each market, futures and chain of options.

4.2.1 update_cboe

Looking at Figure 4.2 we can say that every 15 minutes during the trading hours the Django scheduler sends an event to the Celery application through the Redis message manager asking to execute the update_cboe task. Then celery detaches a new thread and starts executing it in asynchronous way. Within the task, the MongoDB database is contacted to extract the list of all the markets that are part of CBOE and for each of them calls the public API from which is possible to retrieve financial information. In particular, in response you get all the information necessary for the prices and statistical values of the requested market and related futures and options with all available expiration as already explained in section

Table 4.13: Chain history document description

Fields	Description	Type
exchange	Exchange symbol to which the market belongs	String
symbol	Symbol of the market	String
expiration	Symbol of the expiration	String
date	Date and time of the expiration	Datetime
days	Array of day object which contain the date, the list of options and the index position of the last strike price of the day	Array

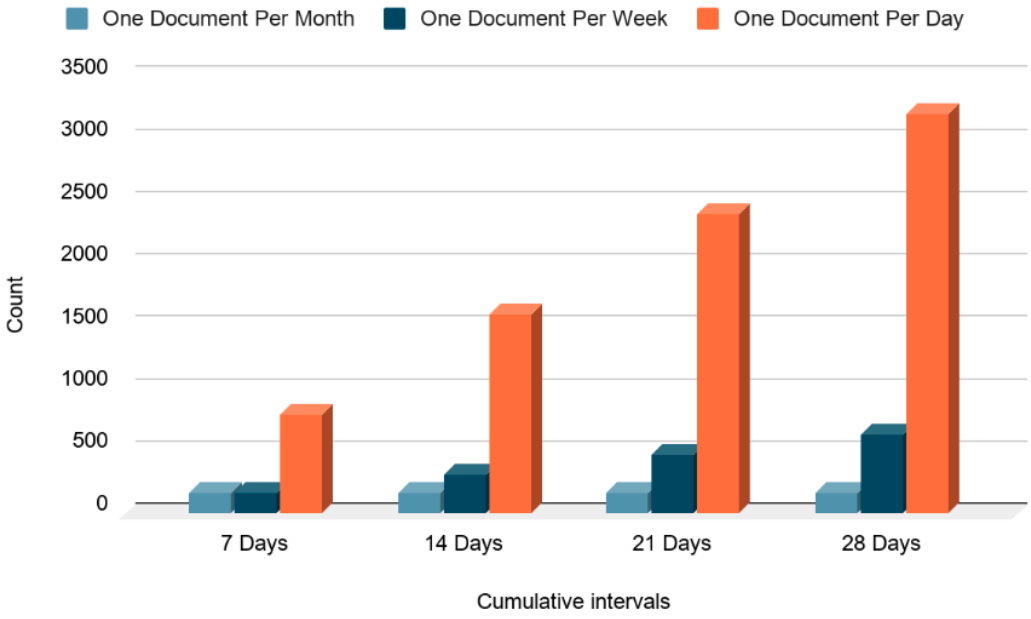


Figure 4.1: Historical data analysis

3.2.2. After which the answer is processed by building the related documents for futures and options and then finally saving them one by one in the database.

4.2.2 update_cme

The first steps are similar to the previous one, also here every 15 minutes during the trading hours, Django ask to Celery for executing the update_cme task asynchronously (Figure 4.3). With the difference that this time the task takes from the database only the markets related to CME and for each of them the public services, already presented in section 3.2.2, are called. In particular, there are two separate

Table 4.14: Historical data: collection total size in MB

Cumulative intervals	Per Month	Per Week	Per Day
7 days	**104,27**	**104,27**	104,41
14 days	**195,48**	195,5	195,77
21 days	**286,7**	286,74	287,13
28 days	**377,91**	377,98	378,49

Table 4.15: Historical data: average document size in MB

Cumulative intervals	Per Month	Per Week	Per Day
7 days	0,66	0,65	**0,082**
14 days	1,23	0,61	**0,082**
21 days	1,79	0,60	**0,082**
28 days	2,36	0,59	**0,082**

APIs to download futures and options prices and two other APIs to obtain statistical information such as volumes and open interests respectively. Within the task these calls are executed sequentially and the corresponding documents are created from each response for finally saving them one by one in the database.

4.2.3 update_eurex

As in the previous tasks, here too the regular interval is 15 minutes after which the task is launched asynchronously (Figure 4.4). Calling up the section 3.2.2 the information from EUREX are not available via REST API, for this reason we have chosen to proceed through web scraping [1] techniques. Analyzing the structure of the website, two distinct paths were identified, one for retrieving information on futures and one for options and these calls are to be made for each market and each available maturity. The documents are constructed by extrapolating the information from the HTML files obtained in response and then being saved individually in the database.

4.2.4 update_history

Unlike the other tasks, here it is not necessary to contact resources external to the application. Indeed, the task interacts only with the database where the updated data at the end of the day are aggregated for each market and option chain into collections dedicated to maintaining historical data (section 4.2).

[1] data extraction from websites

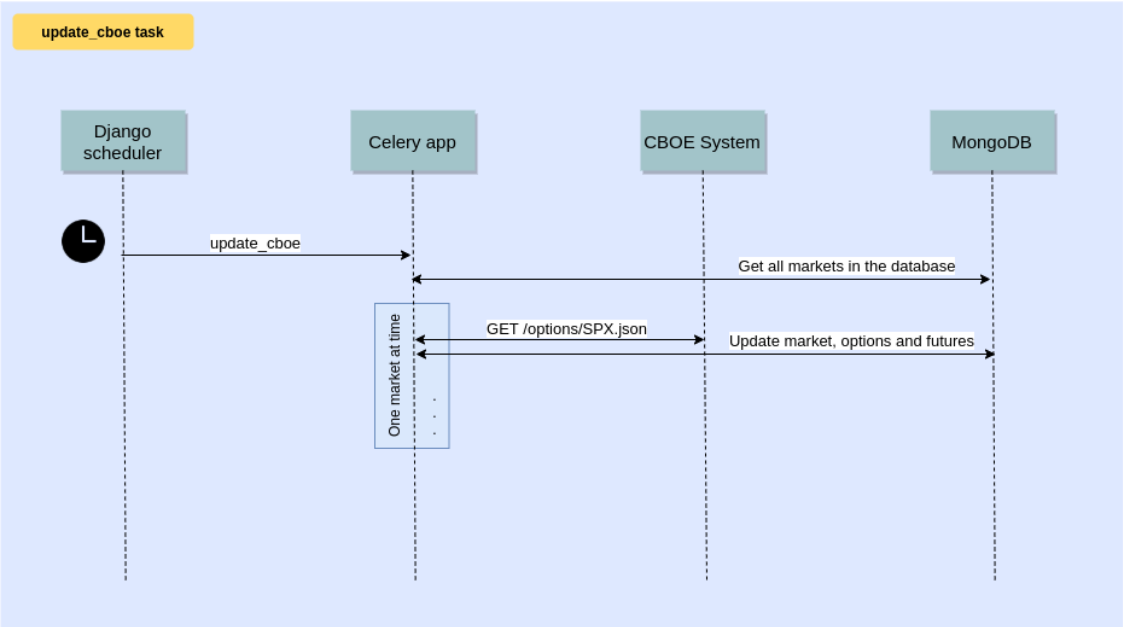


Figure 4.2: Update CBOE async task

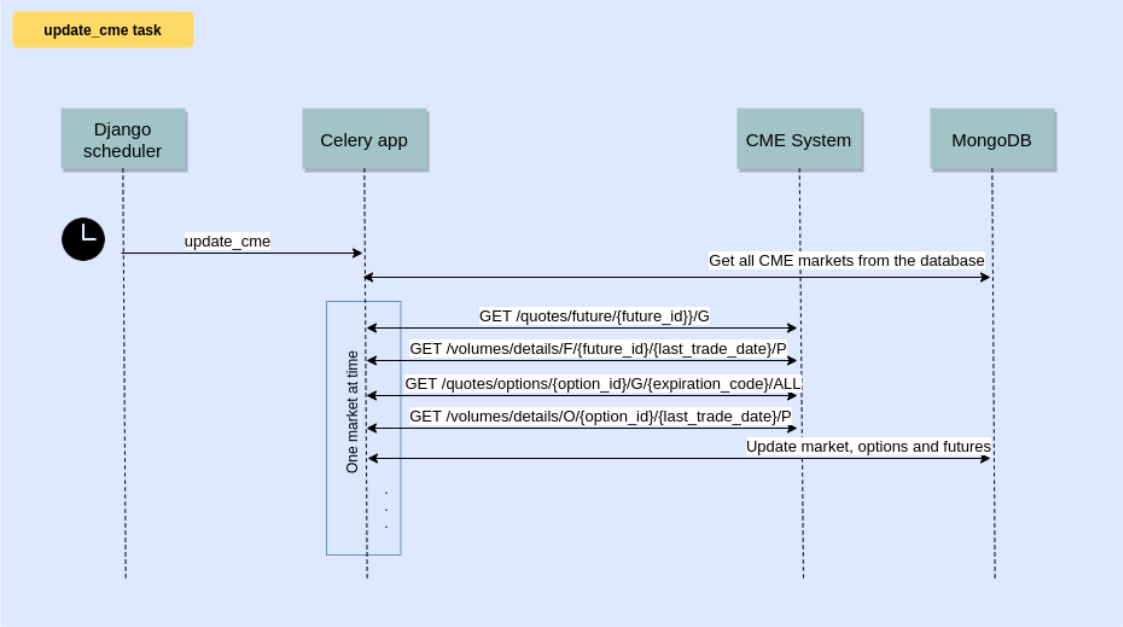


Figure 4.3: Update CME async task

4.3 REST APIs

In this section will deal with the most important web services available that allows the user to operate with the client application and therefore to access the financial

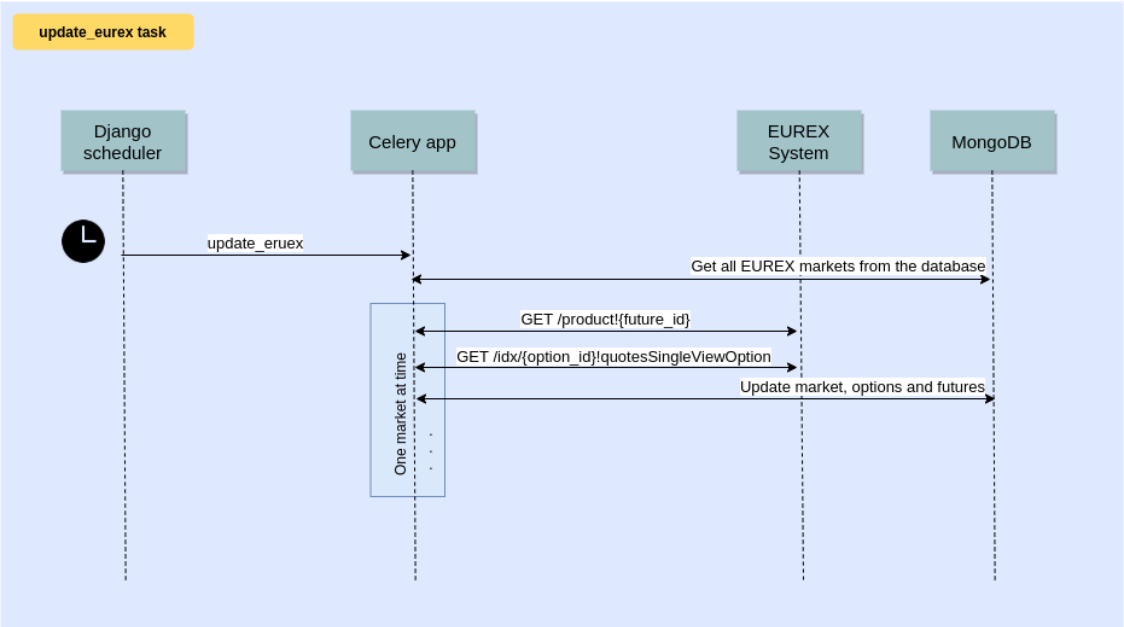


Figure 4.4: Update EUREX async task

Table 4.16: Schedule time for async tasks

TaksId	**Schedule**	**Duration**
update_cboe	Mon-Fri, 8:00-14:00 (America/Chicago)	< 1 minute
update_cme	Mon-Fri, 16:00-18:00 (America/Chicago)	< 10 minute
update_eurex	Mon-Fri, 9:00-19:00 (Europe/Berlino)	< 5 minute
update_history	Mon-Fri, at 20:00 (UTC)	< 30 seconds

data retrieved from the various exchanges. These web services follow the REST guidelines and act as an interface to the following entities: user, market, chain, future, strategy and portfolio. For each of them specific URLs have been defined with different HTTP methods. It is important to note that all defined paths are prefixed by **/api** to distinguish them from other web services that not require an authentication and are not part of the REST API model.

4.3.1 Security

Before starting with the description of each APIs it is important to add that all of them are secured and under HTTPS connection. In fact, every request coming from an unauthenticated or unauthorized client is discarded and an HTTP 401 Unauthorized error is sent in response.

Authentication

The server for user authentication use the default implementation provided by Django Authentication System[2]. In particular, it is a session cookie-based authentication usually consisting of four steps:

1. The client application post an HTTP request to the server with username and password provided by the user
2. The server looks up the username in the database, hashes the supplied login password, and compares it to the previously hashed password in the database. If it is not valid, the access will be denied by sending a HTTP 401 Unauthorized error to the client.
3. If the request is valid, a session ID, which uniquely identifies the user's session, will be created and stored in the database, possibly with an expiration date/time to limit the user's session. After that it will be attached into a response cookie to be returned to the client.
4. For every future client request that require the user authentication, cookies will be attached and the server has only to check if the session ID inside of the cookies is still valid. If so, the access is granted otherwise a new login request is required.

CSRF Protection

Django provides protection against Cross Site Request Forgeries[14] with some easy-to-use features like CSRF middleware and template tag. A CSRF attack occurs when a malicious website contains a link, a form button or some JavaScript that is intended to perform some action on your website, using the credentials of a logged-in user who visits the malicious site in their browser. A related type of attack, 'login CSRF', where an attacking site tricks a user's browser into logging into a site with someone else's credentials, is also covered[3].

4.3.2 User APIs

These APIs are responsible for exposing information related to the user but also to his portfolio and strategies. In fact, as we will see below, there are services that allow the user to create new strategies or delete the irrelevant one but also to carry out read and write operations on them by inserting or deleting market positions.

api/users/

Only the HTTP GET method is allowed and return information of the logged-in user without the need to send in input sensitive information such as username

or email. In fact, after authentication, Django keeps its identifier in session (e.g. username) which remains available for all subsequent HTTP requests by the same user until the end of the session (Listing).

api/users/portfolio/

Only the HTTP GET method is allowed and return data related to logged-in user's portfolio. As already explained, a user can have only one wallet, 1:1 relationship, and for this reason no additional information is required other than the username already present in the session.

api/users/strategies/

Both HTTP GET and POST methods are available. With the GET method you can get a list of strategies belonging to the logged-in user and each element contains information that briefly summarizes a single strategy. In addition, a search string can be passed as query parameter to filter strategies by their name. With the POST method, instead, you can create a new strategy in reference to the logged in user. In particular, when the server receives this request, it checks that in the request body there are valid:

- **groupId** field, the identifier of the market group to which the strategy will refer
- **name** field, the name of the strategy

If the validation process ends without errors then we proceed with the insertion of the new strategy in the database and then return the generated id in response.

api/users/strategies/:id/

Both HTTP GET and POST methods are available. By calling with the GET method the API return data of a single strategy that belong to the logged-in user. Indeed, once the request has been received, the server checks that the id provided in input refers to a strategy existing among those of the user. If so, it returns all the information necessary to view the strategy on the application, otherwise it responds with an HTTP 404 Not Found error. The POST method instead is used to modify a single strategy that belong to the logged-in user. The operations allowed are:

- update the strategy name
- insertion, modification and deletion of one or more positions
- update "whatif" values

For thi The server start first to validate the request body and in case of no errors checks that the id provided refers to an existing strategy among those of the user, after which it proceeds with updating the strategy. If there are any changes to the market positions, additional checks are carried out before proceeding with the saving on the database. In particular, it is necessary to check that the option or future contract for which a position is to be opened exists and belongs to the group of markets in which the strategy can operate, otherwise the positions will not be entered or updated.

api/users/strategies/:id/:chart-id

Per each single strategy is possible to obtain related chart information by calling this HTTP GET method. The chart-id is a path parameter and can be enhanced with **profit** or **greeks** respectively for the strategy Payoff and Greeks information. The API return always an array of JSON object containing data per each single price that the market can assume.

4.3.3 Market APIs

Market APIs only allow read operations and for this reason, all of the following APIs can only be called with the HTTP GET method. In particular, they are mainly dedicated to the client-side display of prices, volumes and open interest but also for the composition of some charts such as historical volatility, open interest by maturity and price history.

api/markets/

Returns the entire list of markets where each element contains few information about them. In addition, a search string can be entered as query parameter to filter markets by their name.

api/markets/:symbol/

Returns all information related to the market symbol passed in input. In particular, in response there are data about:

- market characteristics (e.g. exchange, name, country, currency, etc...)
- last market prices
- all the expiration types and dates for futures and options

api/markets/:symbol/futures/

Returns the entire list of futures for all maturities, belonging to the market symbol passed as input. But not all markets have this information. It depends on whether the underlying is a future or not.

api/markets/:symbol/:chart-id

Per each single market is possible to obtain related chart information by calling this HTTP GET method. The chart-id is a path parameter and can be enhanced with **history**, **volatility** or **open-interest** respectively for obtaining statistics information about the values that the market has assumed in the past days, volatility per each option contract for the next two expiration chains and the sum of option's open interests per each chains expiration. The API return always an array of JSON object containing data per each single price that the market can assume or date.

4.3.4 Chain APIs

As for the markets, Chain APIs are also composed of data reading services only, for this reason, all of the following APIs can only be called with the HTTP GET method. In particular, given a market symbol and a deadline, it is possible to obtain the related list of options but also information processed to obtain some charts, such as open interest for strikes, breakdowns or pressure.

api/chains/:symbol/:expiration/:date/

Returns the data of a single chain identified by the market symbol, type of expiry and expiry date. In particular, the response body will contain:

- characteristics of the chain (e.g. exchange, symbol, market prices)
- list of options
- index indicating the position of the ITM option in the list of options

api/chains/:symbol/:expiration/:date/:chart-id

Per each single options chain is possible to obtain related chart information by calling this HTTP GET method. The chart-id is a path parameter and can be enhanced **volatility/variation**, **open-interest** or **open-interest/cumulative** respectively for obtaining statistics information about the volatility variation per each option contract for the last two days and the sum of option's open interests and the variation for the last two days. The API return always an array of JSON object containing data per each single price that the market can assume or date.

Chapter 5

Client application

This chapter will explain the data management of the client application, the user interface composed by different sections for different functions and operational needs of the user and some examples of use.

5.1 Data management

As previously explained in the section 3.4.2, the client application is based on the React framework and in particular implements the Redux pattern for internal data management using the React Redux library. The key concept of this pattern is to have a single source of data within the application which is commonly called **Store**. This makes the data easily accessible to all parts of the program and avoids duplicate and inconsistent data between the different sections of the application. It also simplifies the development of views where their purpose is only to present the data they take from the store and send internal signals to manage user requests, using functions provided by the library itself.

5.1.1 Data store

From a technical point of view, the store is a JSON Object saved in memory within the browser whose access is allowed only to the application that instantiated it and its life time is strictly linked to the life cycle of the application itself. In our specific case the object is made up of other sub-objects, one for each section that the application presents with the relative data (Listing 5.1). In order to modify the store, the library provides methods for the definition of **Reducer** functions, which accept in input the current state of the store, one **Action** (a JSON Object containing the data that must be saved in the store) and return the updated version of the store. Actions are usually created after an event generated by the user interaction with the view or inside of an asynchronous task. Furthermore, an action must not necessarily be handled inside a Reducer but can also be managed

by an **Effect**, that is a middleware[1] provided by the React Redux library where is possible to put some code for performing some asynchronous operation like an HTTP call to an external API.

```
{
    "app": {
        "mode": "Authenticated",
        "wait": 0,
        "open": false,
        "loading": false,
        "failed": false,
        "groups": [{"symbol": "^GSPC", "name": "S&P 500","type": "index"}, ...],
        "exchanges": [{"symbol": "CBOE", "days": [1,2,3,4,5],"name": ...}, ...],
    },
    "user": {"email": "team@finance.com", "name": ...},
    "modal": {"show": false, "title": ""},
    "markets": {
        "open": true,
        "loading": false,
        "failed": false,
        "search": "",
        "results": [{"symbol": "SPX", "underlying": {"price": 441.67, ...}}, ...],
        "market": {
            "strategies": [],
            "futures": {...},
            "charts": {...},
            "chain": {...},
            "strategy": {..., "whatif": {}}
        },
        "tab": {...},
        "tabs": {...}
    },
    "strategies": {
        "results": [...],
        "charts": [...]
    },
    "portfolio": {
        "currency": "EUR",
        "value": 0,
        "stats": {},
        "chart": {...},
        "strategies": [...],
    }
}
```

Listing 5.1: Example of the application after login store

5.1.2 Data flow

The set of elements that the pattern defines are used to obtain and guarantee a unidirectional flow of data between the views and the store. Figure 5.1 underlines this feature and shows the operation flow after the user's request to retrieve some information of the selected market from the API:

[1]some code that you can put between the framework receiving a request, and the framework generating a response

1. As soon as the user clicks on an element of the market list, an internal event is launched to which an event handler has been associated

2. The event handler intercepts the event and then creates the specific JSON object for the requested action, that is LOAD _MARKET with the market information to be loaded taken from the event itself. Once the object has been created, it will be passed in input to the textbf dispatch function provided by the React Redux library.

3. The action enters the input to the middleware which, based on the type of action, decides to carry out the subsequent operations, in the specific example it will perform an asynchronous HTTP GET call to the API /api/market/:symbol.

4. The call being asynchronous requires two callback functions as input, one to handle the response and the other to handle any exceptions. In both situations, the dispatch function will be used to propagate the server response through an action (e.g. LOAD_MARKET_SUCCESS or LOAD_MARKET_FAILED)

5. The action will pass input to the Reducer together with the latest version of the store and based on the type of the action its status will be updated with the data received from the API.

6. Once the status has been updated, the React framework recreates the view showing the new data.

5.2 User Interface

In a trading application it is important to have a simple, functional and flexible layout with good customization views. From these assumptions the following sections are developed.

5.2.1 Markets

The markets section corresponds to the main section of the application and is divided into two parts (Figure 5.2). On the left there is a panel that gives to the user the possibility to view the list of all available markets and to search for the desired one. On the right, however, once you have selected the market on which you want to operate, a tab will open with all its details. Furthermore, the platform allows you to open multiple markets at the same time by saving them in a list of tabs.

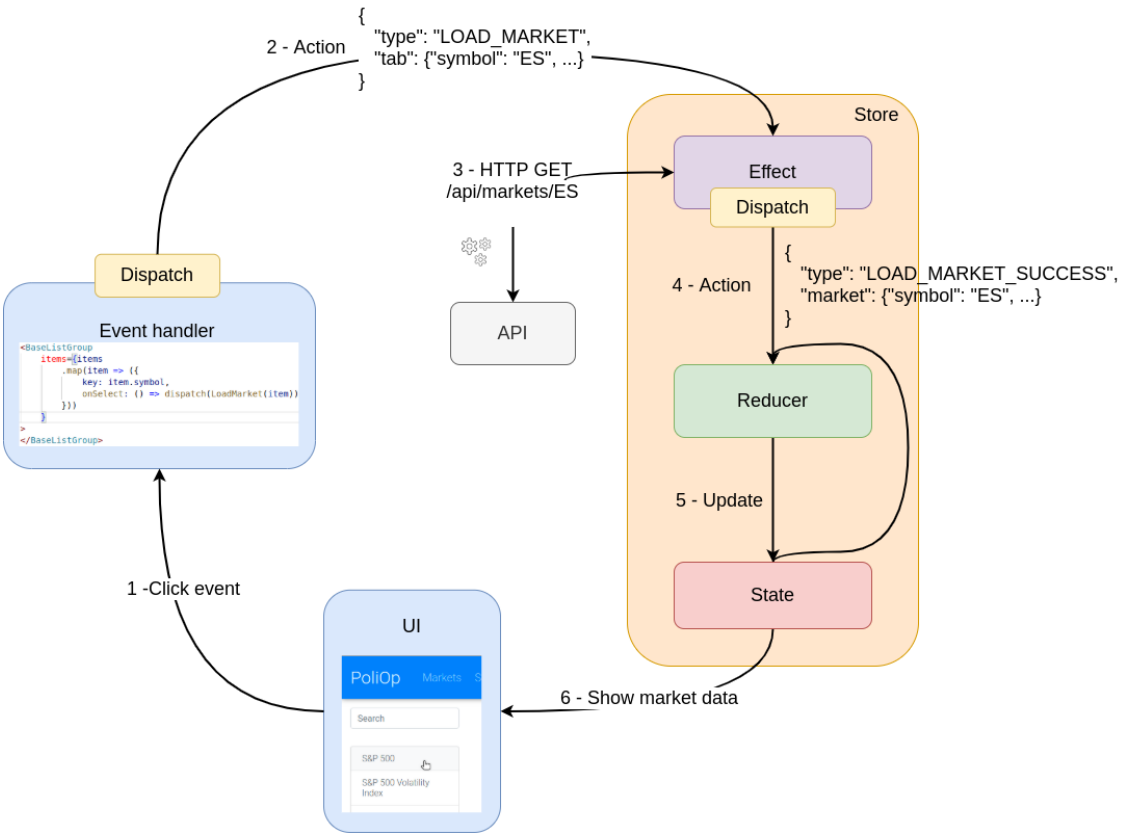


Figure 5.1: Data flow example

Figure 5.2: Markets interface

Tab

Each single tab has a header for the display of information that characterizes the market itself with its latest prices established by the exchange. Subsequently there are four dynamic accordions for different purposes:

- **charts** accordion that contain all visualization and market analysis tools and can be selected via a drop-down menu (Figure 5.3). Each chart provides a zoom in and out, vertical and horizontal scroll with the options to download their content in PNG format, increase or reduce their occupied space on the screen and activate or deactivate their internal grid.

- **positions** accordion, consisting of a drop-down menu to select strategies created by the user for the current market, but also to create a new strategy or modify, update and delete the selected one (Figure 5.4). Furthermore, there are two tables, one for displaying the current opened positions, temporary or not, and one for the closed ones.

- **futures** accordion, that contain a single table that displays all the futures for all available expiration dates with a search bar that allows the user to carry out a textual search 5.5). In addition, from this table it is possible to open a temporary future position by entering the quantity of contracts to buy or sell.

- **chains** accordion, consisting of a drop-down menu to select options for a specific deadline. Then a paged table load and show all the options, both put and call, and the relative prices, volumes and open interest 5.6). From here the user can open a temporary option position by entering the quantity of contracts to buy or sell. Finally, there is a text search bar to filter and search the desired options.

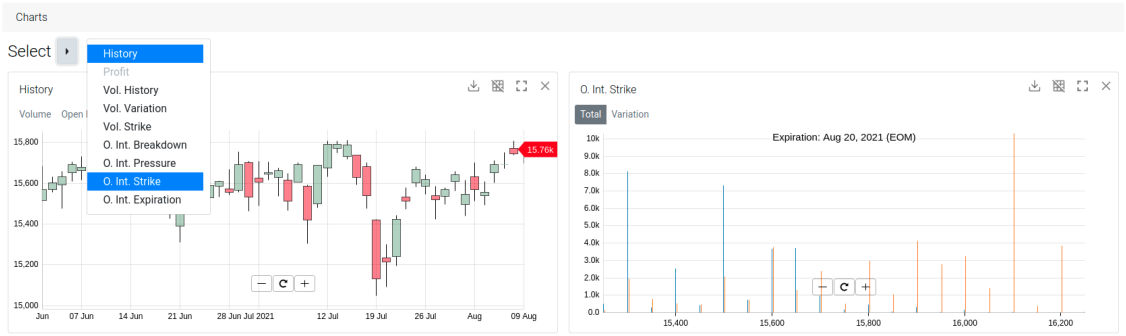


Figure 5.3: Markets charts interface

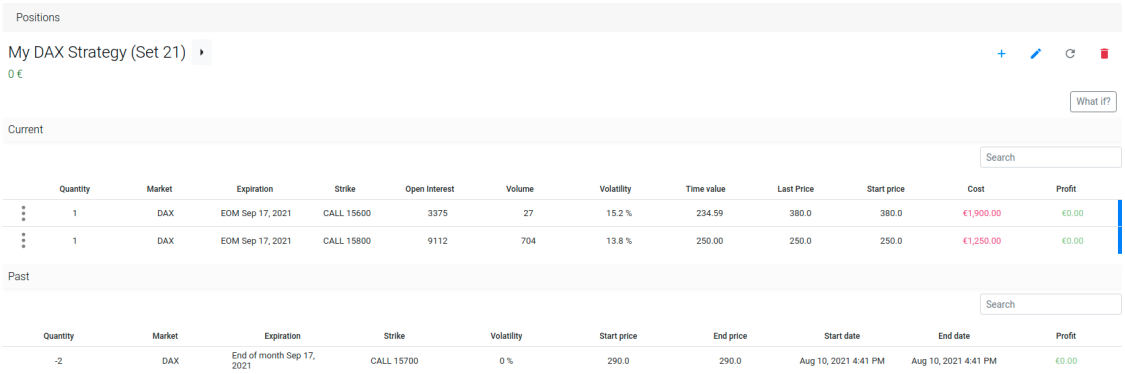


Figure 5.4: Markets positions interface

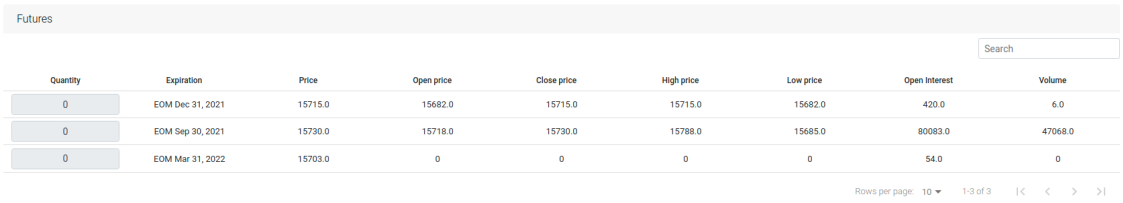


Figure 5.5: Markets futures interface

5.2.2 Strategies

This section is for the management of all strategies created by the user. In particular, it consists of a left panel that shows a list of strategies with the possibility to filter them with a textual research. While on the right there is an initially empty three-column grid that can be filled by selecting strategies from the list. This grid is very useful to view multiple strategies simultaneously and have a brief overview of their trends and profits (Figure 5.7).

Card

A single strategy is showed as a Card element inside of the grid and contains a brief summary on the trend in terms of benefits and costs with three small accordions to view:

- the payoff graph, both at now and at expiration
- the group of markets to which it refers
- the positions currently open

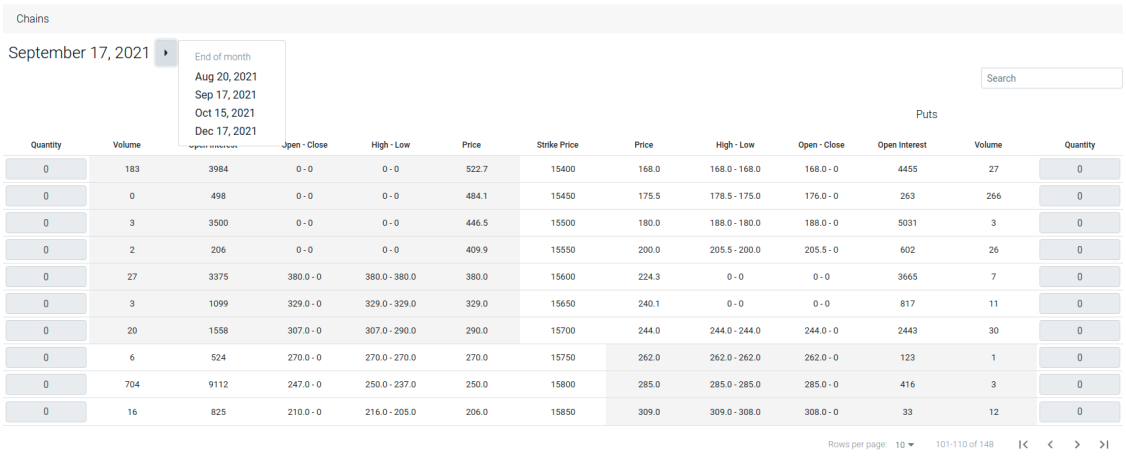


Figure 5.6: Markets chains interface

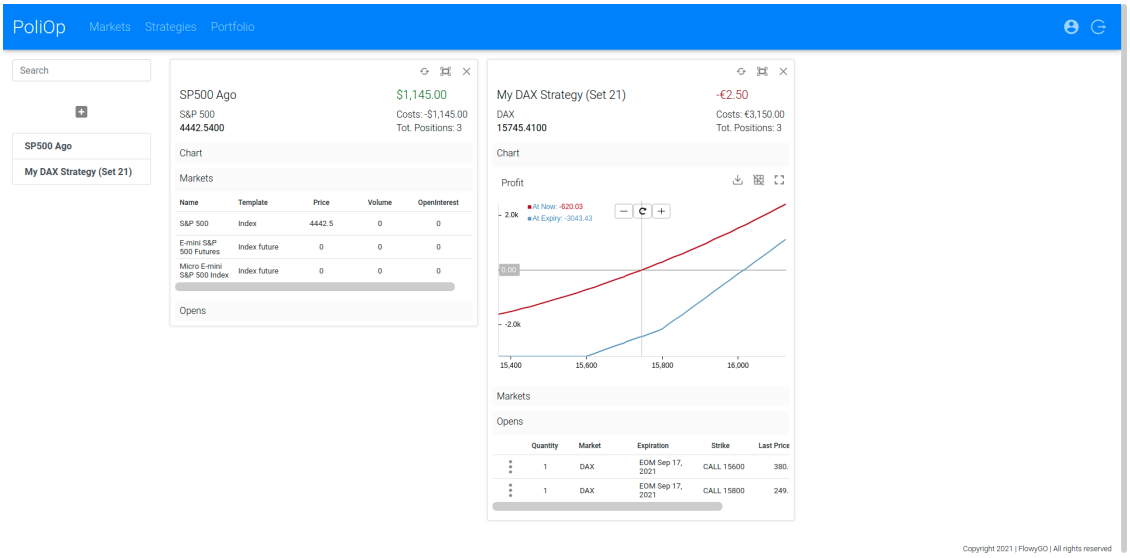


Figure 5.7: Strategies interface

5.2.3 Portfolio

The portfolio is an important section that allows you to view the balance of the user's virtual account but also to graphically represent the trend of his balance over time. This trend is influenced by the strategies closed by the user and summarized in a table, showing their total costs and profits (Figure 5.8).

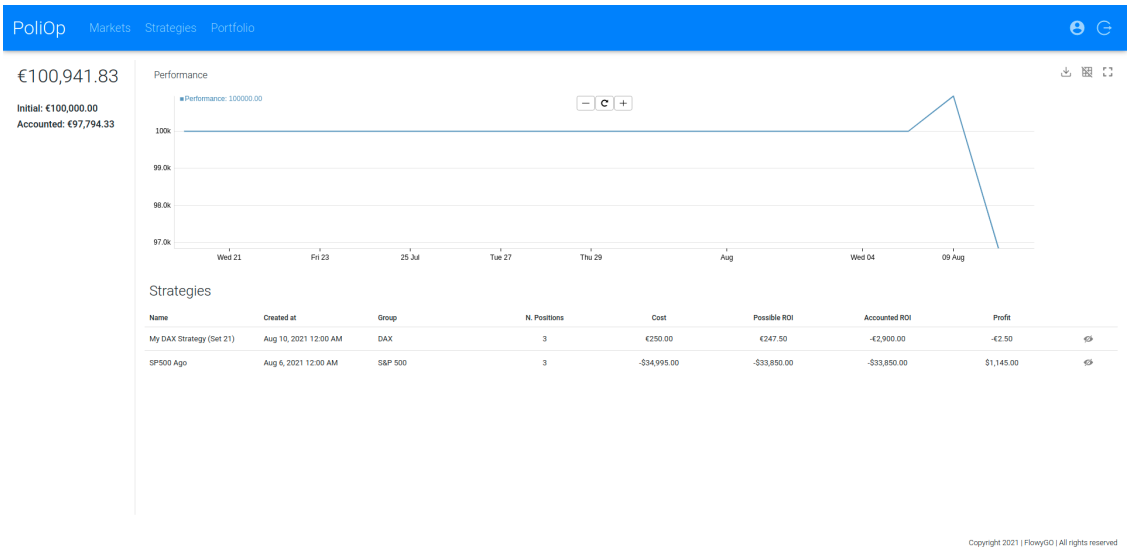


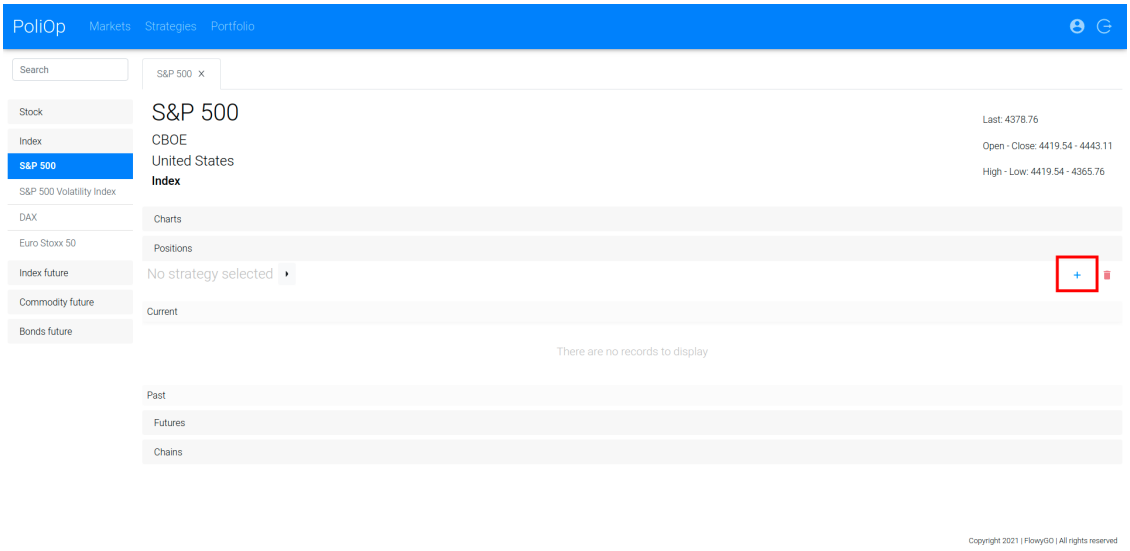
Figure 5.8: Portfolio interface

5.3 Use cases

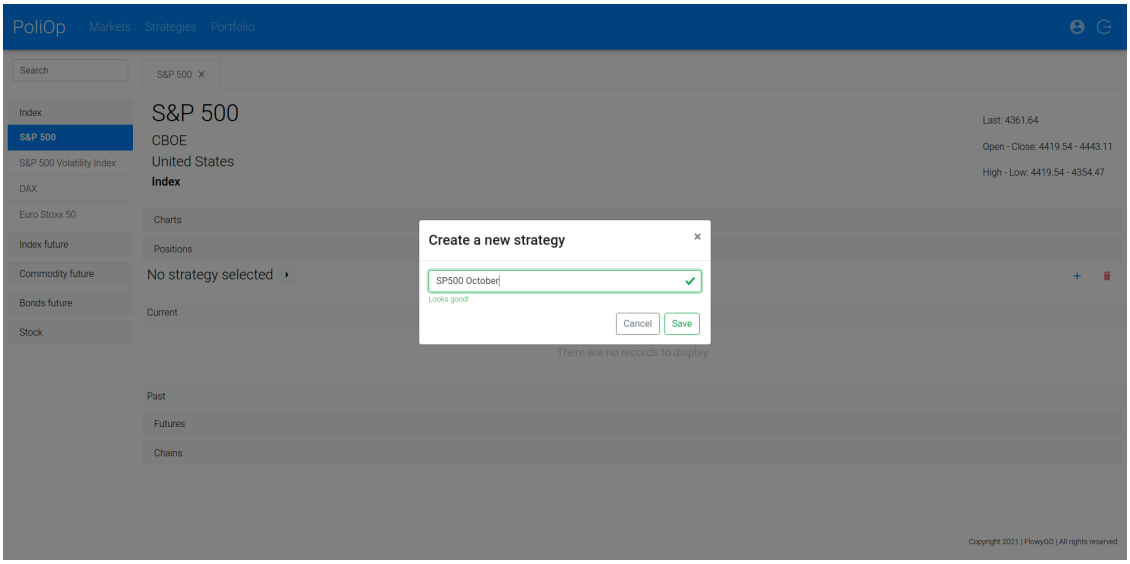
A use case is a set of **scenarios** tied together by a common user goal and is useful to describe the system's behavior under various conditions as it responds to a request. For each use case is required to indicate the **system** being used treated as a black box, the type of user that interacts with the system commonly called **actor** and the functional **goal** that the actor achieves using the system. A single scenario is a sequence of steps describing an interaction between a user and a system. Furthermore it is also required a precondition which must be satisfied before starting the scenario and a post-condition that will be satisfied at the end of it. In this section, some use cases will be presented to describe the various operations that a user can perform such as creating a new strategy, opening new positions within the same strategy (Table 5.1 or 5.2) enter in what-if mode for making some simulation.

Create a new strategy user case show how a user can create its own new strategy. Indeed, after having selected and opened the desired market tab, the user can open the Positions accordion (presented in section 5.2.1), click on the creation button and then enter the name of the new strategy in the modal (Figure 5.9). Once confirmed, the client makes an HTTP POST call to the /users/strategies/ service providing the name of the strategy inserted by the user and the group's identifier of the selected market to link with the strategy. On the server side, the application after having deserialized and validated the request, check if the market group is correct and retrieves the user's username from the active cookie session already

present in the HTTP request and saves the new strategy on the database. If the operation end successfully, a response is sent to the client with the self-generated identifier of the saved strategy. Once the response is received, the client continues its process by making an HTTP GET call to the /users/strategies/:id service to retrieve the information of the identified strategy. Instead, in case of errors during the save process, the client will show an error modal informing the user that the new strategy has not been created.



(a) Open the modal for creating a new strategy



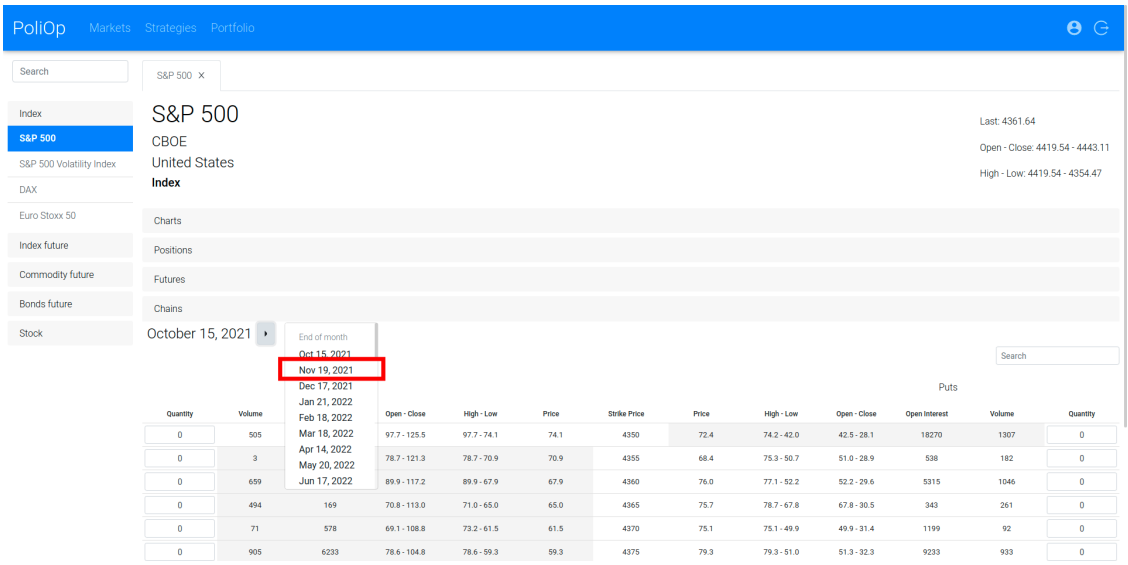
(b) Save the new strategy

Figure 5.9: Creation of a new strategy inside the platform

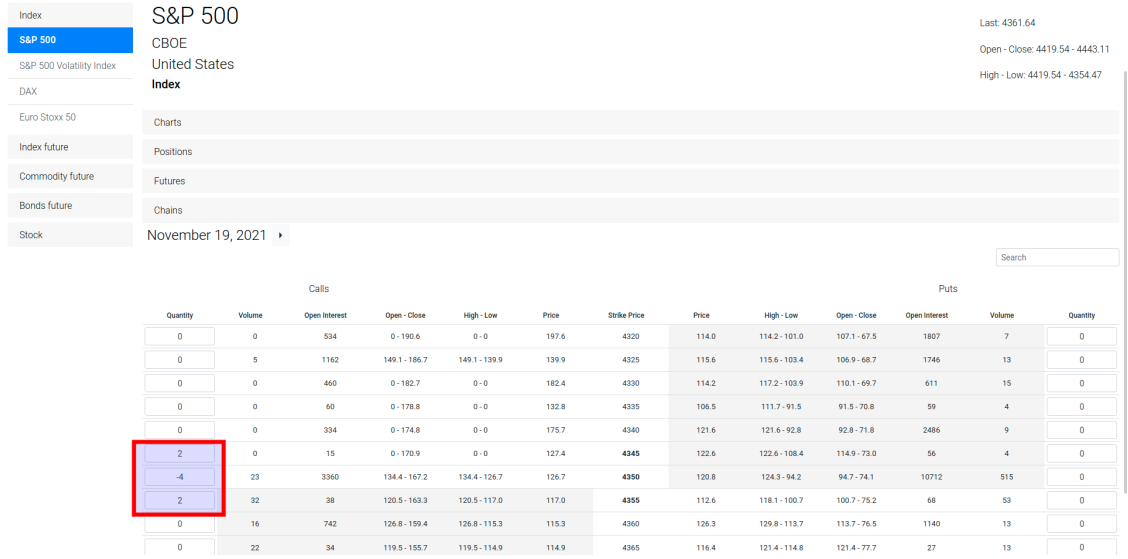
Table 5.1: Use case: create a new strategy

Actors involved	**User**
Precondition	The user U has already authenticated to the system and opened the selected market
Post-condition	The user U has created a new strategy for the market selected
Normal scenario	The user U click on the add button (Figure 5.9a); Insert a valid strategy name on the opened modal; Confirm the creation of the new strategy (Figure 5.9b)
Variant	The user U insert a non valid strategy name and the system does not allow the user to confirm the creation of the strategy

Implement a butterfly spread strategy use case show how a user can add position within an already created strategy in order to simulate the butterfly spread, section 2.5.3. Indeed, after having selected and opened the desired market tab, the user can open the Chains accordion (presented in section 5.2.1), select the desired expiration date and send the right quantity of call and/or put options to buy and/or sell, also called positions (Figure 5.10). Once confirmed, the client makes an HTTP POST call to the /users/strategies/:id service providing the list of positions in request JSON body. The application running on the server, after having deserialized and validated the request, verify if the strategy exist for the user present in the current cookie session and check if all positions are correct by looking the options contract and expiration date. If all the checks are successful, proceed with saving the positions and subsequently send a positive response to the client which reloads the strategy info and showing the new changes. Instead, in case of errors during the save process, the client will show an error modal informing the user that the positions have not been added or updated.



(a) Select the preferred expiration date



(b) Insert the number of contract per options in order to implement the strategy

Figure 5.10: Implement a butterfly spread strategy

Table 5.2: Use case: implement a butterfly spread strategy

Actors involved	**User**
Precondition	The user U has already authenticated to the system and opened the selected market with a new strategy created
Post-condition	The user U has built a butterfly strategy by adding new market positions
Normal scenario	The user U select the expiration from the chain section; Add all necessary market position as temporary Position through the options table; Make all temporary position as open to confirm
Variant	The user U insert zero values and the system does not allow the user to send new position on the strategy

Chapter 6

Conclusions and future works

As already anticipated in the introduction, the objective of the thesis is to create a platform capable of carrying out analysis operations on the derivatives market and implementing operational strategies to exploit the characteristics of futures contracts and options easily accessible via the web browser. At the end of the first version of the platform, we can say that we have reached the main requirements to make it available online and allow new users to use it immediately for their own trading purposes. In particular, after a simple registration, it will be possible to view the main American and European markets available and start operating with them through the personal virtual portfolio, realizing the traditional operational strategies such as Long Call/Put and Short Call/Put but also the most sophisticated ones such as Butterfly or Reverse Butterfly. Furthermore, from a technical point of view, we can confirm that the technologies used are solid foundations to then extend the functionalities with future developments and allow the platform to be distributed through innovative cloud infrastructures.

With regard to future work, there are several fronts that can be developed. In fact, starting from the current state, the internal functions of the application can be extended, such as offering the possibility of automatically generating the best operational strategy based on the latest market conditions. Currently the platform is not suitable for trading intraday as the public APIs provided by CME, CBOE and EUREX update the values every fifteen minutes. These APIs can be replaced with paid ones to obtain data in real time. From the point of view of code and project management we are currently relying on the Git versioning system and the remote server is based on GitHub repository. For this reason it would be interesting easily achievable to create a CI/CD pipeline through the features of GitHub Actions in order to update automatically the Docker image version present online every time a new release is created in the repository.

Bibliography

[1] Celery. Distributed task queue, 2020. URL https://docs.celeryproject.org/en/stable/. [Online; 14 July 2021].

[2] Django. Django authentication system, 2020. URL https://docs.djangoproject.com/en/3.2/topics/auth/default/#. [Online; 14 July 2021].

[3] Django. Django csfr protection, 2020. URL https://docs.djangoproject.com/en/3.2/ref/csrf/. [Online; 14 July 2021].

[4] Docker. Docker overview, 2020. URL https://docs.docker.com/get-started/overview/. [Online; 14 July 2021].

[5] Alessandro Fiori. *Design with MongoDB: best models for applications*. Independently published, Turin, Italy, 2020.

[6] Gordon Scott, Investopedia. Cboe options exchange, 2021. URL https://www.investopedia.com/terms/c/cboe.asp. [Online; 11 July 2021].

[7] James Chen, Investopedia. Chicago mercantile exchange, 2021. URL https://www.investopedia.com/terms/c/cme.aspp. [Online; 11 July 2021].

[8] Rajeev Dhir, Investopedia. Eurex exchange, 2021. URL https://www.investopedia.com/terms/e/eurex.asp. [Online; 11 July 2021].

[9] ISO. Software standards, 2021. URL http://www.iso25000.it/styled-8/. [Online; 11 July 2021].

[10] MongoDB. Introduction to mongodb, 2021. URL https://docs.mongodb.com/manual/introduction/. [Online; 2 August 2021].

[11] Pymongo. Pymongo documentation, 2021. URL https://pymongo.readthedocs.io/en/stable/. [Online; 2 August 2021].

[12] Ragu Ramaswamy. React stockcharts, 2015. URL https://rrag.github.io/react-stockcharts/documentation.html#/overview. [Online; 14 July 2021].

[13] Redis. Redis documentation, 2021. URL `https://redis.io/documentation`. [Online; 14 July 2021].

[14] SquareFree. Cross-site request forgery (csrf), 2020. URL `https://www.squarefree.com/securitytips/web-developers.html#CSRF`. [Online; 14 July 2021].