

**FACULTY
OF MATHEMATICS
AND PHYSICS**
Charles University

BACHELOR THESIS

Daniel Lopata

**Development of a Python Command
Line Interface tool for Cloud Prisma
Margin Estimator**

Department of Software Engineering

Supervisor of the bachelor thesis: Mgr. Petr Škoda, Ph.D.

Study programme: Computer Science

Prague 2025

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In date

Author's signature

I would like to thank my family for their support throughout the studies and during the writing of this thesis. I would also like to thank my supervisor, Mgr. Petr Škoda, Ph.D., for his guidance and invaluable advice, which were crucial for me and helped me enormously. Last but not least, I would like to thank my consultants from Deutsche Boerse Group, Adam Ruzicka and Miroslav Paris, for their time, professional insights, welcoming approach, and the opportunity to work on a real-world project.

Title: Development of a Python Command Line Interface tool for Cloud Prisma Margin Estimator

Author: Daniel Lopata

Department: Department of Software Engineering

Supervisor: Mgr. Petr Škoda, Ph.D., Department of Software Engineering

Abstract: Managing financial risk is a critical task in trading and clearing processes. Margin estimators are integral to this process, as they calculate the collateral required to cover potential losses from market movements. Cloud Prisma Margin Estimator is a solution developed by Deutsche Boerse Group (DBG) to calculate the margin requirements for financial portfolios. DBG wanted a tool that would provide traders with the functionality of the company's estimator. The focus of this thesis was to develop a command-line tool that would enable the users to use the estimator's API. In the thesis, we describe the requirements, design, implementation, and provide the documentations for our tool. The part of the requirements is to deliver comprehensive tutorials for the tool's features and detailed tutorial on the Calendar spread strategy.

Keywords: Command Line Interface, Cloud Prisma Margin Estimator, Python, API integration

Název práce: Vývoj nástroje příkazové řádky v Pythonu pro Cloud Prisma Margin Estimator

Autor: Daniel Lopata

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Petr Škoda, Ph.D., Katedra softwarového inženýrství

Abstrakt: Řízení finančních rizik je v procesech obchodování a zúčtování klíčovým úkolem. Odhad marže je nedílnou součástí tohoto procesu, protože vypočítává kolaterál potřebný k pokrytí potenciálních ztrát z pohybů na trhu. Cloud Prisma Margin Estimator je řešení vyvinuté skupinou Deutsche Boerse Group (DBG) pro výpočet maržových požadavků pro finanční portfolia. DBG chtěla nástroj, který by obchodníkům poskytoval funkce firemního estimátoru. Těžištěm této práce bylo vyvinout nástroj příkazového řádku, který by uživatelům umožnil používat API estimátoru. V práci popisujeme požadavky, návrh, implementaci a poskytujeme dokumentaci k nástroji. Součástí požadavků je dodat komplexní návody k funkcím nástroje a podrobný návod na strategii Calendar spread.

Klíčová slova: příkazová řádka, Cloud Prisma Margin Estimator, Python, API integrace

Contents

Introduction	9
1 Cloud Prisma Margin Estimator	10
1.1 API design and key endpoints	10
1.1.1 Key endpoints	10
1.2 Existing usages	13
1.2.1 Graphical User Interface (GUI) on the web	13
1.2.2 REST API	14
1.2.3 Swaggerhub GUI	14
1.3 Conclusion	15
2 Requirements analysis	16
2.1 Stakeholders and their requirements	16
2.2 Specification	17
2.2.1 System definition	17
2.2.2 Functional requirements	17
2.2.3 Non-functional requirements	19
2.2.4 Tutorials	19
3 Design	21
3.1 Architectural approaches	21
3.1.1 Monolithic architecture - single script	21
3.1.2 Layered architecture	21
3.1.3 Modular monolith	22
3.1.4 Conclusion	23
3.2 Architecture	23
3.2.1 Software system layer	23
3.2.2 Container layer	24
3.2.3 Component layer	24
3.3 Programming language and libraries	25
3.3.1 Programming language	25
3.3.2 Libraries	26
4 Developer documentation	29
4.1 Development tools	29
4.2 CLI implementation	30
4.2.1 CLI structure	30
4.2.2 Commands	30
4.2.3 Handler creation	31
4.2.4 Command argument validation	31
4.2.5 Extending the CLI	31
4.2.6 CLI flow	31
4.3 Application layer implementation	32
4.3.1 Interface for request handlers	32
4.3.2 Exporting of data	33

4.3.3	Using the cpme_api client	33
4.3.4	Error handling and logging	33
4.3.5	Environment variables	34
4.3.6	Extending the application layer	34
4.3.7	Application layer flow	35
4.4	Tool structure	35
5	Testing	37
5.1	Test coverage	37
5.2	Mocking	37
5.3	User testing	38
6	User documentation	39
6.1	Installation and setup	39
6.1.1	Installation	39
6.1.2	Setup of the .env file	40
6.1.3	Using Jupyter Notebook for tutorials	41
6.2	Showing all available commands	41
6.3	Fetching available products	42
6.4	Fetching series for products	43
6.5	Fetching live snapshots	43
6.6	Fetching snapshots	44
6.7	Sending a portfolio to the estimator	44
6.8	Calculating and aggregating the margins	45
6.9	Limitations and unsupported features	45
6.9.1	OTC portfolio support	45
6.9.2	Historic data request delays	46
6.10	Tutorials	46
7	Use case - calendar spread strategy	47
7.1	Key terms	47
7.2	Definition of a calendar spread	48
7.2.1	Long calendar spreads	48
7.3	Margin and calendar spreads	50
7.3.1	Margin treatment in calendar spreads	50
7.3.2	Assignment risk and margin changes	50
7.3.3	Summary of margin benefits	50
7.4	Tutorial for calendar spread	51
	Conclusion	52
	Bibliography	53
	List of Figures	56
	List of Tables	57

Introduction

Financial markets allow individuals and institutions to trade various types of assets, including stocks, bonds, and derivatives. Stocks represent ownership of a portion of a company, giving the holder a claim to part of the company's assets and earnings [1]. Bonds are a type of investment where individuals lend money to a company for a fixed period, receiving interest payments in return [2]. Derivatives are financial contracts whose value depends on the performance of an underlying asset, such as a stock, bond, or commodity [3].

Financial markets primarily trade two types of derivative instruments. The first of these are Exchange Traded Derivatives (ETDs), which are financial contracts traded on regulated exchanges, such as futures and options. An option gives the buyer the right, but not the obligation, to buy or sell an asset at a specific price at any time during the life of the contract. A futures contract obligates the buyer to purchase a specific asset, and the seller to sell and deliver that asset at a specific future date. [4] The second of these instruments are Over-the-Counter (OTC) interest rate swaps. OTCs are customized financial contracts traded directly between parties, outside of exchanges. [5] Because of their different trading environments, ETDs and OTCs follow different margining rules and risk management practices. ETDs are typically subject to centralized clearing and require initial margins set by a clearing house ¹ to manage risk. In contrast, OTC contracts often involve bilateral agreements where margin requirements can be negotiated.

These trades on markets are made easier by brokers, intermediaries that execute buy and sell orders on behalf of traders. In many cases, traders do not purchase assets outright but instead use borrowed funds to increase their market exposure. This practice introduces credit risk, which arises when a trader is unable to repay the borrowed funds. For example, imagine a trader opens a leveraged position in a volatile stock using borrowed capital. If the stock price suddenly drops, the value of the trader's position may fall below the borrowed amount. If the losses exceed the margin the trader initially deposited, the broker faces a risk of not recovering the full amount, especially if the trader cannot quickly provide additional funds. To mitigate this risk, brokers require traders to deposit a margin.

Margin is the collateral an investor must deposit with a broker or exchange to cover the credit risk associated with borrowing funds or entering into contracts. It acts as a buffer for potential losses, ensuring that the investor can fulfill their obligations in the event of trade problems. While leverage amplifies both gains and losses, the margin serves as a risk management tool for the broker, ensuring they are partially protected even if the position turns against the trader. Access to margin trading is typically not granted to every investor automatically; brokers often assess the trader's financial situation, experience, and risk profile before allowing leveraged trading. [6]

Margin trading allows investors to control a larger position than they could with their own capital alone. By borrowing funds from the broker, traders can

¹A financial institution which facilitates the exchange of payments, securities, or derivatives transactions

amplify their potential returns, as gains are calculated on the total position size rather than just their initial investment. However, as we have said, this leverage also magnifies losses. If losses exceed the margin, a margin call may be triggered, thus requiring additional funds. The broker demands additional funds to restore the account balance. If the investor cannot satisfy the margin call, the broker can close out some of investor's positions to bring the account into balance. [7] This means that the investor is in the loss. To maximize potential profit while minimizing risk, it is important to set the margin high enough to avoid forced liquidation but also low enough to maintain sufficient leverage.

Portfolio margin is a risk-based approach that considers the overall risk of a portfolio. It allows for lower margin requirements for so-called hedged positions. The hedging from a financial point-of-view means to hold two or more open positions while trading. The investor can then possibly offset losses from the first position with the gains from the second one, as broker calculates requirements based on net risk, not individual positions. [8] It is a regulatory-based method to assess risk. [9] This approach frees up capital by aligning margin requirements with the portfolio's risk profile. In simpler terms, since the risk of the entire portfolio is lower due to hedging strategies, the trader does not need to deposit as much margin as they would for unhedged positions. This allows traders to use the extra available capital for other investments or trading opportunities, improving capital efficiency. [10]

For effective trading, investors seek to optimize their margin requirements based on their portfolio's risk profile. The solution to this challenge is the use of margin estimators, which help determine the optimal margin necessary for maintaining positions while minimizing unnecessary capital allocation.

Margin estimation plays a crucial part in managing risk in financial markets. It involves calculating the amount of collateral required to cover potential losses when trading with either borrowed funds or entering into derivative contracts. This concept is crucial for managing risk and ensuring investors meet their financial obligations. [6]

Accurately calculating margin requirements presents several challenges, particularly for complex portfolios, but with the advancement of computational methods, margin estimation has become more precise and efficient.

The availability of free margin estimators is vital for individual investors and traders, especially those who do not have institutional backing or advanced financial tools but still participate in margin trading. By providing access to margin calculators, individuals can better manage their investments and mitigate risks associated with margin trading.

The Cloud Prisma Margin Estimator is a solution developed by Deutsche Boerse Group (DBG) to provide precise and scalable margin calculations.² Prisma benefits clearing members and traders by reducing capital costs while ensuring regulatory compliance.

The goal of this thesis was to develop a command line interface (CLI) tool that provides investors with a convenient way to calculate their margins and utilize the functions of the Cloud Prisma Margin Estimator. The thesis focuses on integrating the CLI tool with the Cloud Prisma Margin Estimator, ensuring

²<https://www.eurex.com/ec-en/services/margincalculators/eurexotc-clear-prisma-margincalculator/cloud-prisma-margin-estimator>

its usability and efficiency. This project was carried out in collaboration with DBG. An important part of the work also includes writing comprehensive tutorials for the functionality supported by the tool, along with a detailed guide for the Calendar spread strategy. These tutorials are intended to help investors fully leverage the tool's capabilities, but they are omitted from the thesis for brevity.

Now let us describe the structure of our thesis. In chapter 1 we describe the Cloud Prisma Margin Estimator and analyse its usages. After that, we analyse our requirements in chapter 2. Next, we introduce the design of our application and reasons which drove our decisions (chapter 3). In chapter 4 we describe the implementation and tools which we used during the development. Afterwards, we provide information about testing (chapter 5) and then we outline how to use the tool in chapter 6. In chapter 7 we describe the Calendar spread strategy, and its relation to margin trading, and we provide a tutorial on how to utilize our tool to implement it. In the end we conclude our work and discuss achieved results.

Tools such as Grammarly, Writefull, and ChatGPT were used while writing this thesis to correct grammatical mistakes and improve the writing style.

1 Cloud Prisma Margin Estimator

The Cloud Prisma Margin Estimator (CPME) is a cloud-based tool offered by Eurex Clearing ¹ for simulating margin requirements under their Prisma margin methodology. ² It provides several benefits for risk management. It offers transparency and accuracy in margin calculations. The CPME aligns with Eurex Clearing's production systems for Exchange Traded Derivatives (ETDs). Additionally, the CPME offers high accuracy for Over-the-Counter (OTC) interest rate swaps. An interest rate swap is a forward contract in which one stream of future interest payments is exchanged for another based on a specified principal amount. [11]

Its setup ensures that users always have access to the latest version of Prisma, benefiting from continuous updates and new features without needing to install the software locally. Users can simulate margin requirements for single positions, which refer to individual investments or trades they hold in a financial market, or for entire portfolios, considering all of their combined investments. By assessing margin requirements in advance, traders can better manage their risk exposure, ensuring they have sufficient collateral to cover potential losses. This proactive approach helps prevent unexpected margin calls, reduces the risk of forced liquidations, and enables more informed decision-making in portfolio management.

1.1 API design and key endpoints

The Cloud Prisma Margin Estimator provides a RESTful (Representational State Transfer) API for submitting portfolios and retrieving margin estimates. The API is designed for scalability, security, and ease of integration into automated risk management systems. It follows standard RESTful principles, using JSON for data exchange and HTTP methods for interactions. The API supports authentication mechanisms such as API keys to ensure secure access. The API documentation is available on SwaggerHub ³, together with JSON schemas.

1.1.1 Key endpoints

The CPME API includes several endpoints for margin estimation, retrieving results, querying supported products, and fetching historical series of margin requirements. For our usage, only the following endpoints are important.

Portfolio estimation endpoint (`/estimator`)

Method: POST

Purpose: Submits a portfolio for margin estimation.

¹<https://www.eurex.com/ec-en/find/about-us>

²<https://www.eurex.com/ec-en/services/margining/eurex-clearing-prisma>

³<https://app.swaggerhub.com/apis-docs/dbgservice/cPME/2.0>

Request: A JSON payload ⁴ containing position details and optional valuation parameters.

Response: A JSON object with margin estimates, including initial margin and drill-downs⁵ for series.

Products endpoint (/products)

Method: GET

Purpose: Fetches a list of exchange-traded products supported by the margin estimator.

Response: A list of financial instruments and their margin parameters.

Example response with only a single product listed for clarity:

```
{
  "business_date": 20180809,
  "live": false,
  "live_timestamp": 0,
  "products": [
    {
      "product": "OSMI",
      "instrument_type": "option",
      "clearing_house": "EUXCDEFF",
      "prod_name": "OPT ON SWISS MARKET INDEX",
      "prod_isin": "CH0008616382",
      "underlying_isin": "CH0009980894",
      "currency": "CHF",
      "product_type": "OINX",
      "extended_product_type": null,
      "margin_style_flag": "T",
      "exercise_style_flag": "E",
      "product_settlement_type": "CASH",
      "final_settlement_time": "09:00",
      "product_tick_size": 0.1,
      "product_tick_value": 1,
      "liquidation_group": "PEQ01",
      "xm_eligibility": false
    }
    ...
  ]
}
```

Series endpoint (/series)

Method: GET

Purpose: Fetches a list of series⁶ for specified exchange-traded products.

⁴https://app.swaggerhub.com/apis-docs/dbgservice/cPME/2.0#/default/post_estimator

⁵Drill-downs are a more detailed view for series

⁶<https://www.investopedia.com/terms/o/optionseries.asp>

Response: A list of series together with additional information.

Example response with only a single series listed for clarity:

```
{
  "business_date": 20180809,
  "live": false,
  "live_timestamp": 0,
  "list_series": [
    {
      "product_id": "OGBL",
      "contract_date": 20190621,
      "contract_maturity": 201906,
      "expiry_maturity": 201906,
      "call_put_flag": "C",
      "exercise_price": 160,
      "version_number": "0",
      "iid": 18249016
    }
    ...
  ]
}
```

Live snapshots endpoint (/live_snapshots)

Method: GET

Purpose: Fetches and provides a list of real-time margin snapshots for a given business day generated at different times throughout the day.

Response: A list of live snapshots for the given business day.

Example response:

```
{
  "snapshots": [
    {
      "business_date": 20180809,
      "live": true,
      "otc_available": true,
      "cash_available": false,
      "live_timestamp": 0
    },
    {
      "business_date": 20180809,
      "live": true,
      "otc_available": true,
      "cash_available": false,
      "live_timestamp": 1562162876
    }
  ]
}
```

Snapshots endpoint (/snapshots)

Method: GET

Purpose: Fetches and provides an overview of available margin snapshots for specific dates, helping users determine which snapshots can be used for further margin estimations or analyses.

Response: A list of end-of-day or first live snapshots.

Example response:

```
{
  "snapshots": [
    {
      "business_date": 20180809,
      "live": false,
      "otc_available": true,
      "cash_available": false
    },
    {
      "business_date": 20180810,
      "live": true,
      "otc_available": true,
      "cash_available": false
    }
  ]
}
```

1.2 Existing usages

In this section, we will discuss existing usages of the CPME. We will also address the advantages and shortcomings of all the options.

1.2.1 Graphical User Interface (GUI) on the web

The CPME is accessible via a user-friendly GUI ⁷ seen in Figure 1.1, allowing users to input data and simulate margin requirements easily.



Figure 1.1 GUI on the web

Through the GUI, users can enter individual ETD positions or upload portfolios in various formats, such as CSV files or Eurex Clearing member reports. ⁸

⁷<https://cpme.eurex.com/>

⁸<https://deutsche-boerse-risk.github.io/CloudPrismaMarginEstimator/docs/gui.html>

The GUI supports margin calculations for Eurex-listed derivatives (ETDs), OTC interest rate swaps excluding FX swaps, and single-ISIN repo trades, a repurchase agreement where a single security, identified by its International Securities Identification Number (ISIN), serves as collateral for the transaction.⁹ This interface provides immediate results, making it suitable for both basic and advanced users who need to assess margin requirements.

However, the GUI has some limitations. It mainly focuses on margin calculations and does not support other functions of the estimator, which its API could access, like generating products and series or fetching the snapshots. It only supports the estimator endpoint. Additionally, the output options are limited, namely, it is available in limited form in Excel, which can restrict users who require more detailed reports.

1.2.2 REST API

For more advanced or technically adept users, the CPME offers integration via a RESTful API, which is documented on SwaggerHub¹⁰, where users can see supported endpoints and needed schemas. This API allows good integration with client systems and processes, enabling automation and flexibility in margin simulations. Users can upload portfolios in CSV format, just like in web GUI, and the API supports calculations for ETDs, OTC interest rate swaps, and single-ISIN repos. The API is particularly useful for large-scale operations or when integrating margin simulations into existing risk management systems. It is accessible after obtaining an API key from DBG Digital Business Platform¹¹.

However, the API has its drawbacks. Mainly, it is not accessible for users who are not technically adept, as it requires a good understanding of RESTful APIs and programming. Additionally, users need to handle authentication, request formatting, and error handling correctly, which can add complexity to the implementation.

1.2.3 Swaggerhub GUI

The CPME also offers a web-based interface via SwaggerHub¹², providing interactive documentation for its RESTful API. Users can explore available endpoints, submit test requests, and review response structures in a user-friendly environment. This GUI enables API integration by allowing developers to understand request parameters, response formats, and error-handling mechanisms before implementation.

This GUI also has its drawbacks. Firstly, the SwaggerHub page does not provide options for exporting data. Secondly, uploading portfolios directly for the estimator endpoint is not possible, which can complicate the workflow for some users. This limitation means users must handle data preparation and transfer outside of SwaggerHub before interacting with the API.

⁹<https://deutsche-boerse-risk.github.io/CloudPrismaMarginEstimator/docs/gui.html>

¹⁰<https://app.swaggerhub.com/apis-docs/dbgservice/cPME/2.0>

¹¹<https://console.developer.deutsche-boerse.com/>

¹²https://app.swaggerhub.com/apis-docs/dbgservice/cPME/2.0#/default/get_products

1.3 Conclusion

CPME provides multiple access options—GUI, RESTful API, and SwaggerHub—so that the users can choose the most suitable approach based on their technical expertise and needs. While the GUI offers a straightforward and intuitive experience, the REST API provides automation and integration capabilities for more advanced users.

However, each method has its limitations, such as restricted output options or the restriction of available endpoints in the GUI, the technical complexity of API integration, or the inability to upload one’s portfolio to SwaggerHub. To address these issues, the development of a dedicated CLI tool becomes important.

The CLI tool bridges the gap between ease of use and automation, offering the flexibility of the API while maintaining accessibility for non-technical users. It enables easy interaction with the CPME, allowing efficient portfolio uploads and result exports without requiring deep API knowledge. Since it mitigates many shortcomings of listed access options to CPME and offers many advantages—automation, integration, usability, export, and robustness—the CLI tool presents itself as a practical solution for risk managers, traders, and analysts who seek a robust yet user-friendly way to interact with the CPME.

Feature	GUI	REST API	SwaggerHub
Ease of Use	High	Low	Medium
Automation	No	Yes	Limited
Integration with Systems	No	Yes	No
Exporting Results	Limited	Yes	No
Multiple Endpoints	No	Yes	Yes
API Key	No	Yes	Yes

Table 1.1 Comparison of CPME Access Methods

2 Requirements analysis

The requirements analysis was done in collaboration with Deutsche Boerse Group (DBG), the university's partner company. There were multiple meetings with representatives from DBG, during which the requirements were gathered. Various ideas regarding the usage of the CPME API were discussed. In the end, we decided to develop a Python CLI tool that uses a Deutsche Boerse-provided CPME API client. At first, we came up with the idea of a simple tool for sending requests to a single endpoint with an aggregation of results and their export. This idea was later improved and extended so that multiple endpoints could be accessed from the tool depending on the command, and the export of returned data was also enhanced. More arguments were also added so that the user would be able to modify the request. We try to address all of the shortcomings mentioned in the previous chapter for the current usages of CPME. In this chapter, we will first introduce the stakeholders and what they expect from our tool, and then we will describe the specification.

2.1 Stakeholders and their requirements

In this section, we will describe the stakeholders and their expectations of our tool. We will define three groups of stakeholders - users of the tool, developers, and CPME experts from the company.

Users of the tool

The user base is expected to consist of both technical and non-technical users, which makes clarity and ease of use especially important.

- Easily usable and understandable interface
- Multiple available endpoints
- Wide selection of request modifications
- Multiple choices of export types

Developers

Developers from the company are responsible for maintaining and extending the tool in the future. Since the tool is intended to evolve in the future, the code should be designed with maintainability and extensibility in mind.

- Well-written, well-documented and tested code
- Good design and ease of extension

CPME experts

This group includes financial experts from the company with a deep understanding of the CPME system and its real-world applications. Their role is to ensure that the tool can correctly handle relevant use cases.

- Tool capable of handling complex use cases from a financial point of view
- Well-written tutorials which showcase the usage of the tool so that users can easily start using it

2.2 Specification

In this section, we will first define our system and then talk about the specifications for our tool. This includes functional and non-functional requirements. Functional requirements are the features or operations of a product that developers need to incorporate to allow users to perform their tasks effectively. Nonfunctional requirements are not related to the functionality of the system, they describe how the system should behave and perform. [12] These requirements stem from issues from Deutsche Boerse internal GitHub and numerous meetings with the company's representatives.

2.2.1 System definition

The system is a CLI tool developed to interact with Deutsche Boerse's Cloud Prisma Margin Estimator. It enables users to retrieve and process margin data using the estimator's RESTful API. It provides functionalities such as exporting filtered data, fetching data regarding margin snapshots, and uploading ETD portfolios.

2.2.2 Functional requirements

Let us now list the functional requirements for our system. They are denoted as CLI for CLI functionality.

FR-1-CLI Exporting filtered products data

The user shall be able to export filtered product data. The user can export data in CSV, Excel, or JSON format, specify an output directory, and apply filters so that only relevant products are included. The exported file shall follow a predefined naming convention in the form of *DATE_VERSION_products* and contain all product data columns, including extension fields.

FR-2-CLI Exporting filtered series data

The user shall be able to export filtered series data. The user needs to specify the products for which the series shall be listed. The user can export data in CSV, Excel, or JSON format, specify an output directory, and apply filters to include only relevant series. The exported file shall follow a predefined naming

convention in the form of *DATE_VERSION_series* and contain all series data columns, including extension fields.

FR-3-CLI Fetching live snapshots

The user shall be able to retrieve and display available live snapshots on a given date. The live snapshots include timestamps, snapshot numbers, and relevant metadata. The output shall be formatted for easy readability and displayed in the console.

FR-4-CLI Fetching end-of-day snapshots

The user shall be able to retrieve and display available end-of-day snapshots for a given range of dates. The snapshot data shall include relevant metadata. The output shall be formatted for easy readability and displayed in the console.

FR-5-CLI Uploading an ETD portfolio

The user shall be able to upload an ETD portfolio from a CSV file. The user must specify a file path, and the system shall validate the file format before sending the portfolio to the estimator for processing. Upon successful upload, the portfolio shall be processed, and results shall be generated in CSV, Excel, or JSON format.

FR-6-CLI Generating template from series for ETD portfolio

The user shall be able to generate an ETD portfolio file based on filtered series data. The user can trigger this functionality using a flag while generating series. The generated file shall follow the structure of the ETD portfolio. Users can manually define position sizes, and the file shall be exportable in CSV format.

FR-7-CLI Uploading ETD Portfolio in an ETDPortfolioInner data model

The user shall be able to upload an ETD portfolio from a CSV file in ETD-PortfolioInner data model from the API client. The user must specify a file path, and the system shall validate the file format before sending the portfolio to the estimator for processing. Upon successful upload, the portfolio shall be processed, and results shall be generated in CSV, Excel, or JSON format.

FR-8-CLI Uploading ETD portfolio for date intervals and aggregate margins and export in the form of graph

The user shall be able to upload an ETD portfolio from a CSV file to aggregate margins and export in Excel and in graph for a date interval. The user must specify a file path, and the system shall validate the file format before sending the portfolio to the estimator for processing. Upon successful upload, the portfolio shall be processed for each business date in the interval, and results shall be generated in Excel and graph.

2.2.3 Non-functional requirements

Let us now list the non-functional requirements for our system. They are denoted as TEST for testing, DOC for documentation, and INST for installation.

NFR-1-TEST Unit test implementation

The system shall include comprehensive unit tests for all critical components of the CLI functionality, ensuring correctness, reliability, and maintainability.

NFR-2-DOC Auto-generated code documentation implementation

The system shall have extensive documentation in the form of comments. It shall be possible to automatically generate documentation for the tool's CLI commands, functions, and classes.

NFR-3-INST Installation and entry point

The system shall provide a working pyproject.toml file with the installation of the package and dependencies and to set up the entry points for the execution of the tool.

2.2.4 Tutorials

Besides the functional and non-functional requirements, the system shall also provide tutorials in Jupyter Notebook. These tutorials shall provide users with a demonstration of how to use the tool.

TUTORIAL-1 Tutorial for products

The tutorial in a Jupyter Notebook should demonstrate the usage of the products CLI feature. The tutorial shall include examples of usage with some input parameters and outputs with descriptions.

TUTORIAL-2 Tutorial for series

The tutorial in a Jupyter Notebook should demonstrate the usage of the series CLI feature. The tutorial shall include examples of usage with some input parameters and outputs with descriptions.

TUTORIAL-3 Tutorial for live snapshots

The tutorial in a Jupyter Notebook should demonstrate the usage of the live snapshots CLI feature. The tutorial shall include examples of usage with some input parameters and outputs with descriptions.

TUTORIAL-4 Tutorial for end-of-day snapshots

The tutorial in a Jupyter Notebook should demonstrate the usage of the snapshots CLI feature. The tutorial shall include examples of usage with some input parameters and outputs with descriptions.

TUTORIAL-5 Tutorial for estimator

The tutorial in a Jupyter Notebook should demonstrate the usage of the portfolio upload CLI feature. The tutorial shall include examples of usage with some input parameters and outputs with descriptions.

TUTORIAL-6 Tutorial for date interval margin calculation and aggregation

The tutorial in a Jupyter Notebook should demonstrate the usage of the date interval margin calculation CLI feature. The tutorial shall include examples of usage with some input parameters and outputs with descriptions.

TUTORIAL-7 Tutorial for Calendar spread strategy

The tutorial in a Jupyter Notebook should demonstrate the complex usage of CLI features. The tutorial shall describe the strategy and showcase the way how the tool can help to implement it, making it a practical resource for users interested in calendar spreads and margin analysis.

3 Design

After listing the requirements for our tool in the previous chapter, we now look at the design of our system. First, we discuss the approaches which were considered for the architecture and then describe the chosen one. After that, we explain the choice for programming language and libraries used in the development and implementation.

3.1 Architectural approaches

The choice of the architecture for a software system is a very important task. Good architecture makes the system easy to understand, easy to develop, and easy to maintain [13]. On the other hand, bad architecture might negatively impact the system and slow down its development.

The core objectives of our software architecture and design that we look at are:

- Modularity - a design where the system is broken into independent, self-contained modules, each responsible for a specific part of the functionality.
- Maintainability - a design that is ready for an implementation of change requests, such as adding new features. [14]

We want our architecture to be simple enough so that the system is easy to develop and does not introduce many layers of unneeded abstraction, but at the same time to be easily extensible and with a good separation of concerns. We also want it to be flexible to adapt to new requirements easily. The listed principles guide us through the selection of our architectural pattern from several available options.

3.1.1 Monolithic architecture - single script

In this approach, all of the functionality is in a single script. It is a very simple approach to build a CLI application, as all of the logic is put together. Usually, it is used for small utilities, where the speed of development is taken into account.

For our purpose, however, it is not good enough. The main drawbacks of this architecture are its hard maintainability as the tool grows, bad separation of concerns, and low modularity. The drawbacks go against the principles that we stated earlier, so we decided not to use this architecture.

3.1.2 Layered architecture

In a layered architecture, the tool is divided into distinct layers. It is partitioned technically, meaning based on the technical role or type of responsibility each part of the code fulfils, and driven by layered cohesion. Each layer in this architecture has its distinct role, and the layers are usually stacked on top of one another, as responsibilities are organized into logical horizontal layers. Modules are defined within these layers. [15] The layers may vary, but common layers in a three-level architecture include:

- Presentation Layer - also called first tier, handles user input and user interface, in our case this would be CLI parsing
- Business Logic Layer - also called second tier, contains the main functionality and core business domain, includes services and business logic, in our case this would be the processing of user data and making API calls
- Data Access Layer - also called the third tier, manages data persistence and retrieval, like reading from a database, in our case not needed [16]

LAYERED ARCHITECTURE

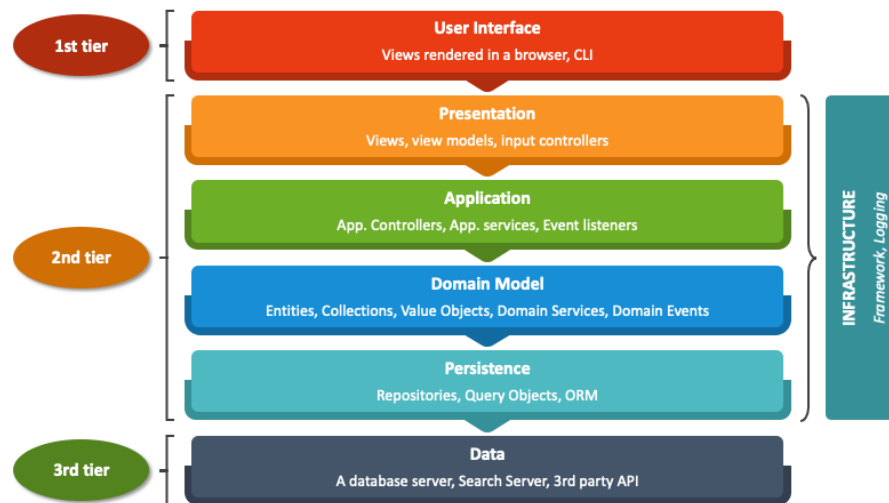


Figure 3.1 Layered Architecture[16]

This architecture has its advantages. One of its main advantages is its simplicity, as it is easy to implement. The cost of development is also not very high and projects with this architecture can be developed and delivered quickly. This is because the pattern is well known. [15] It promotes a good separation of concerns, as it divides the application into distinct layers. This is achieved because components within a specific layer deal only with logic that belongs to that layer. ¹

Apart from these good features, this architecture also offers some drawbacks. One of the main disadvantages of this architecture is its low modularity, as the components inside a layer are typically tightly coupled and designed to work together. [15] Layers may also put some rigidity on the codebase, as these layers usually should not be bypassed. Multiple layers might also be harder to extend, and there could be higher coupling between the layers and the components.

Because of these drawbacks, mainly low modularity and unwanted rigidity, we decided not to use this architecture.

3.1.3 Modular monolith

A modular monolith architecture structures an application as a single deployable unit while internally organizing the codebase into well-defined, feature-based

¹<https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

modules. [17] It organizes the codebase based on a functional perspective based on domains instead of a technical perspective. Responsibilities are organized into vertical layers by logic, and those vertical layers might then be organized into technical layers. The modules are then defined inside these layers. [15]

To implement such an architecture in a CLI context, the command pattern can be used effectively. The command pattern encapsulates each command from the CLI as a separate class. In this design, each command is represented by an object which implements a common interface. [18] This makes it easy to modify, add, and extend the commands, as the logic of each command is encapsulated in its class. It uses object-oriented principles, making its design naturally promote modularity, and it provides good separation of concerns as well. This makes it suitable for multi-functional CLI tools.

One of its main advantages is its extensibility. New commands can easily be added by creating a new class that implements a common interface. Another advantage is its maintainability, as it divides the code into modules with clear boundaries. Each command can be developed and tested independently, so the development complexity is reduced.

This architecture also has a drawback. It might contain more code and more complexity than a simple monolithic architecture since each command requires its own class. This could make this architecture seem a little excessive for simple tools, but as our tool needs to support multiple endpoints, this added complexity is justified.

3.1.4 Conclusion

After reviewing the advantages and disadvantages, we come to the conclusion to use the modular monolith architecture implemented by the command pattern. Its modularity and extensibility align with the principles we stated earlier and with the requirements for our tool, mainly the support for multiple endpoints. We believe that this architecture will help to keep the codebase clean, easily maintainable, and well-organized.

3.2 Architecture

After our discussion on architectural approaches, let us now design the architecture of the tool. We will use the modular monolith architecture implemented with a command pattern, as stated in the previous section. We will describe the architecture using the C4 model ², with a focus on the first three levels - software system, containers and components. A software system is made up of an arbitrary number of containers, which in turn consist of an arbitrary number of components.

3.2.1 Software system layer

In this layer, we describe the interactions between users of our system, and potential interactions between our system and external systems. As we can see in Figure 3.2, in our case, our system interacts with an external Python client

²<https://c4model.com/>

provided by Deutsche Boerse Group ³, called `cpme_api`, for making the actual API calls to the estimator API.



Figure 3.2 C4 model - Software system layer

3.2.2 Container layer

Our software system is divided into two distinct containers, as can be seen in Figure 3.3:

- CLI: This container allows the user to interact with the system by entering commands and options. It also processes user input, validates it, and forwards to our application layer for handling.
- Application layer: The application layer container takes care of the business logic of the system. It is divided into distinct commands. It processes the input received from the CLI, makes the requests through an external client and exports the results.

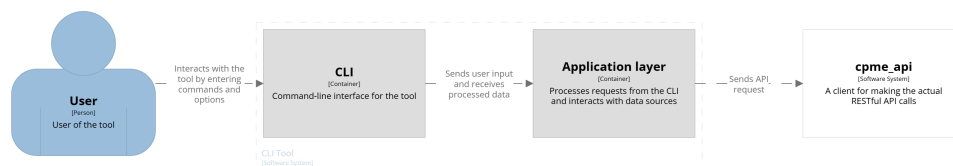


Figure 3.3 C4 model - Container layer

3.2.3 Component layer

Let us now take a look at the components in our respective containers. In Figure 3.4, we can see all of the components for the CLI container, and in Figure 3.5, we can see the components for the application layer container.

³https://github.com/ky573/CloudMarginEstimatorClient/tree/gk101_estimator_demo_tool_update/cpme_api

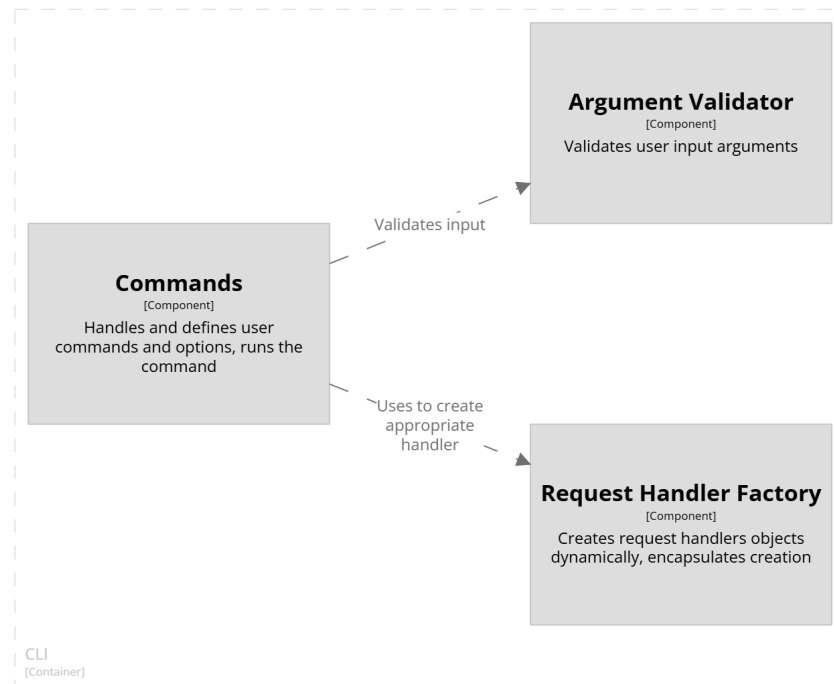


Figure 3.4 C4 model - CLI components

3.3 Programming language and libraries

In this section, we will discuss the programming language chosen for our tool, and also libraries that support its development.

3.3.1 Programming language

This CLI tool is developed in Python ⁴, a general-purpose programming language which is known for its simplicity, readability, and ease of integration with external services. ⁵ It is also currently the most popular programming language. ⁶ Python was selected because of the following reasons:

- Ease of integration with cpme_api client provided by Deutsche Boerse Group
- Extensive choice of libraries, making our development easier and faster
- High level of abstraction, which makes the codebase easier to maintain

⁴<https://www.python.org/>

⁵<https://www.geeksforgeeks.org/python-language-advantages-applications/>

⁶<https://www.tiobe.com/tiobe-index/>

3.3.2 Libraries

For our tool, we want to choose widely used libraries that are easy to understand and easy to master. By using these libraries, we want to make the development easier for ourselves so that we do not need to implement everything. Our libraries are listed below.

cpme_api

As was said before, we will be using the Python client provided by Deutsche Boerse Group to make API calls to the estimator API. It provides wrappers for our requests and also models for our data. ⁷

Click

Click is a library for creating command line interfaces. It is open-source, and it is constantly improved. It was chosen because it is quite simple to use, yet at the same time powerful enough to create complex CLI applications with support for subcommands. It also supports working with file paths from the command line, and validating input types.

Argparse, Python's native library, was also considered, but the features of Click were found to be more powerful.

Pandas

Pandas is an open-source library used for data analysis and data manipulation. It provides rich functionalities for manipulating, reading, and writing data in various formats. It also provides seamless integration with our other libraries, like OpenPyXL. Integration with Jupyter Notebooks is also very easy, which is nice for our tutorials.

Another alternative for data processing was NumPy, but it does not provide such a good level of abstraction and data processing.

OpenPyXL

OpenPyXL is a library for reading and writing Excel files. It supports many features, like creating sheets, structured format, and formatting of data. We selected it to support the export of data into Excel format. It can also be used easily with Pandas.

Plotly

Plotly is a library for creating plots in Python. It supports a wide variety of plots. We selected it to provide plotting of the graph of initial margins for our margin calculation.

⁷https://github.com/ky573/CloudMarginEstimatorClient/tree/gk101_estimator_demo_tool_update/cpme_api/models

Kaleido

Kaleido is a cross-platform library for generating static images. It is used together with Plotly to export the graph as a static image.

Requests

Requests is a library used to make API calls, parse responses, and handle errors. The latter two features are the reasons we selected it. It is quite simple to use and handles HTTP requests very well.

Another alternative which was considered, `http.client` from the standard Python library, lacks the user-friendly abstraction of Requests.

Python dot-env

Python dot-env is a library that loads environment variables from a predefined `.env` file. It allows storing information like API keys outside of the source code, making the codebase more secure and flexible. It reduces hardcoded values in the code. It was selected to store the API key for API usage, and to optionally store a proxy address for routing in case of need. A proxy might be useful because it acts as an intermediary between the client and external services. Additionally, proxies are helpful in bypassing network restrictions, and that might be needed for some users.⁸

An alternative was to load the API key manually from a configuration file, but Python dot-env provides a more flexible and standardized way of configuration.

⁸<https://answers.microsoft.com/en-us/windows/forum/all/what-is-a-proxy-address-and-how-do-i-find-mine/1afa2546-6373-e011-8dfc-68b599b31bf5>

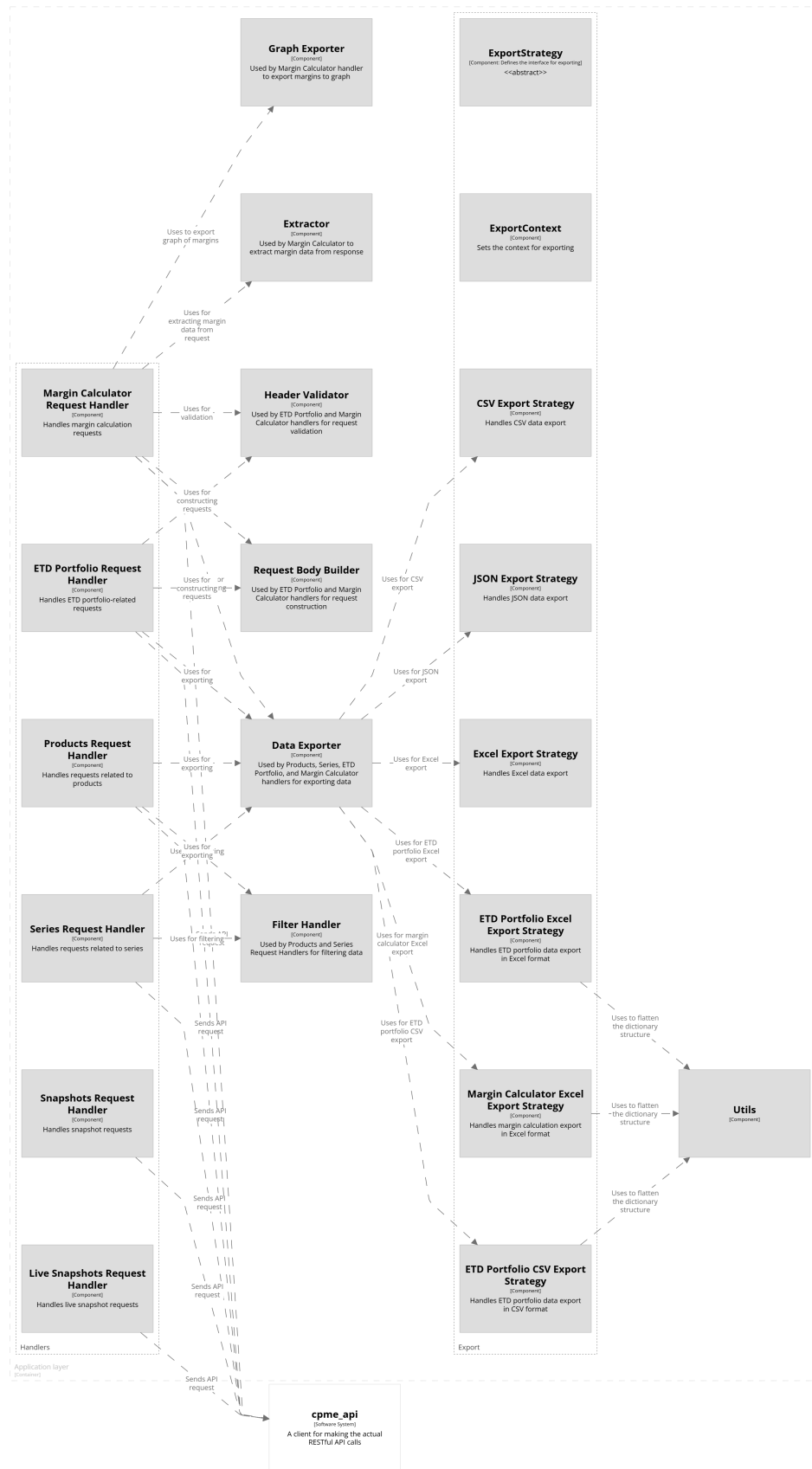


Figure 3.5 C4 model - Application layer components

4 Developer documentation

After discussing the design of our tool in the previous chapter, we now look at the implementation. First we describe the tools that we used for our development. Then, we explain the implementations of the command-line interface and the application layer.

4.1 Development tools

In this section we describe the tools that were used during the development of our command-line tool. They are not part of the core implementation, but they provide code quality improvement, code testing and version control.

GitHub Issues

GitHub Issues were used to break down requirements and commit linking¹. Each issue was labelled and categorized to improve transparency and organization.

GitHub Actions

GitHub Actions were used to set up the pipeline for continuous integration (CI). Continuous integration means that every change that is committed triggers an automated build and test.² GitHub Actions were selected for CI because of ease of use and seamless integration with GitHub.

Pylint

Pylint is a static code analyzer for Python. It is used to check code quality, improve coding standards, and it can make suggestions on how the code should be refactored. It was chosen because of its popularity and compliance to PEP-8³, which is the Python coding standard.

Black

Using a code formatter can improve code quality, maintainability, and collaboration among developers, and it ensures that the codebase remains consistent. [19] Black is an automatic code formatter. It was used with basic settings to reformat the code according to PEP-8. It was selected because of its strict style.

MyPy

MyPy is a static type checker for Python. It was used because we wanted to check type annotations and ensure type safety across the codebase. It also

¹After the relocation of the repository from internal GitHub to public one, the mapping of commits to issues might be incorrect.

²<https://aws.amazon.com/devops/continuous-integration/#:~:text=Continuous%20integration%20refers%20to%20the,for%20a%20release%20to%20production.>

³<https://peps.python.org/pep-0008/>

improved code quality and type hints, thus making code much easier to understand and modify.

Pytest

In order to check the correctness of our code, we want to have tests implemented. A good suite of tests also makes the code easier to refactor. Tests are easier to implement with a testing framework. Pytest was selected due to its features, such as parametrized tests, simple assertion handling, and fixtures.

Sphinx

One of our non-functional requirements required to have thorough code documentation so that it could be generated automatically by a tool. We chose Sphinx as our tool to generate the documentation from docstrings in the code. Sphinx enabled the generation of documentation into HTML form, and the documentation could then be easily maintained and automated.

4.2 CLI implementation

The CLI implementation represents the interface through which the user communicates with our tool. This section describes its structure, key commands, and functionality.

4.2.1 CLI structure

CLI was implemented using Click library. We offer multiple commands in our tool, which are grouped into a main `click.group`. Individual commands are implemented as functions and are registered to this group. It allows for flexible extension by adding new commands without modifying the core logic.

4.2.2 Commands

We provide the following commands for our tool:

- **get_products:** Fetches and filters product data based on user-defined criteria, exporting them in a desired format.
- **get_series:** Fetches and filters series data based on user-defined criteria, exporting them in a desired format.
- **get_live_snapshots:** Fetches and prints out data about live snapshots for the given business day.
- **get_snapshots:** Fetches and prints out data about snapshots for the given range of business days.
- **etd_portfolio:** Fetches the margin data for given portfolio, exporting them in desired format.

- **margin_calculator**: Fetches the margin data for the given portfolio, aggregates the initial margins, and exports the result in the desired format.
- **version**: Prints out the version of the tool.

4.2.3 Handler creation

In order to encapsulate the creation of handlers for endpoints, the tool implements a Simple Factory. The simple factory is a programming idiom. It describes a class with a creation method that chooses the instantiated object based on a parameter [20]. The appropriate handler for a command is selected based on user input. Each handler class encapsulates the logic for processing the command's arguments, fetching API data, and exporting the results.

4.2.4 Command argument validation

We want to ensure that our tool behaves correctly. Click library offers some default validators that were used in our validation logic. These include managing required parameters and parameter types. For the validation of dates and paths of export directories, however, we implemented our own validators. It was because we did not like the error messages of built-in validation. Each handler thus has its own validator, and those validators are implemented using the Template method pattern. The template method defines an algorithm and lets the subclasses implement its steps. [21] If the user passes wrong arguments, he gets informed about it with some meaningful error message either from our validators or Click default validators.

4.2.5 Extending the CLI

If we want to extend the CLI by implementing new commands, we need to:

1. Define a new command function and register it to the main `click.group`.
2. Implement a corresponding handler class to process the command's arguments and logic.
3. Register the handler in the endpoint handler factory.
4. Implement validation class if needed.

4.2.6 CLI flow

Figure 4.1 illustrates the request processing flow in the CLI. It is described using the UML flow diagram. A full black node is an initial node, arrows show the flow control, a rounded rectangle is an action, a hexagon means a decision node, and a full circle inside another circle is an end of the workflow.

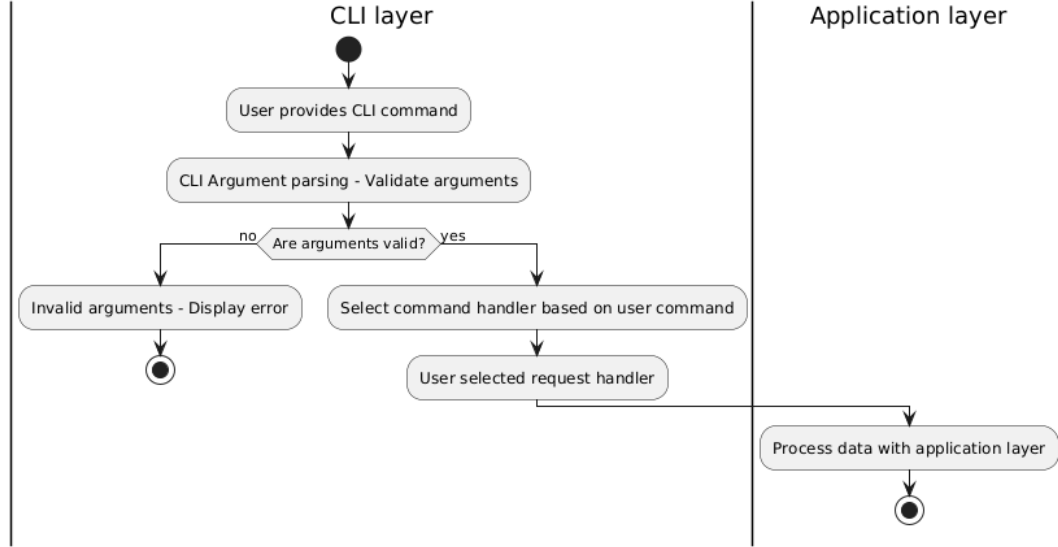


Figure 4.1 CLI processing flow

4.3 Application layer implementation

The implementation of the application layer handles the core logic of our tool. It communicates with the external `cpme_api` client to make API calls, processes and filters the data, and handles the export of the data. The application layer consists of several components, each responsible for handling a specific functionality. The key components include:

- **Request Handlers:** Classes that implement the logic for different CLI commands, such as fetching products, series, or estimator data.
- **Export Strategy:** Exporting is implemented using a strategy pattern. The strategy pattern defines a family of algorithms, while encapsulating each one, making them interchangeable. [21] This pattern allows us to decide the export during the runtime. The tool supports the export of data in various formats, namely CSV, Excel, and JSON.

In this section, we describe the structure of the application layer and its responsibilities.

4.3.1 Interface for request handlers

All of our request handlers inherit from a single base class⁴ and implement its interface. This ensures consistency across all handlers, and it allows easy integration with the CLI. Each handler processes input arguments from the user, fetches a response from the client, applies filters if needed, and then provides the output in the desired form.

Listing 4.1 Interface for Request Handlers

⁴https://github.com/ky573/CloudMarginEstimatorClient/blob/gk101_estimator_demo_tool_update/margin_estimator_tool/src/core/request_handler_base.py

```

class RequestHandler(ABC):
    """Class to handle sending requests to the API."""

    @abstractmethod
    def process_and_provide_output(self) -> None:
        """Method for processing a request and exporting data;
        implemented by subclasses."""

```

We chose this interface to enforce a contract for all request handlers. The `process_and_provide_output` encapsulates the sending of the request and post-processing logic such as filtering and exporting results.

4.3.2 Exporting of data

The tool uses a strategy pattern for data exporting. This allows adding different exporting options without modifying the existing ones, adhering to the open-close principle. Our export strategies once again implement a common interface for handling the exporting logic.⁵ All potential new strategies need to implement it.

Listing 4.2 Abstract Interface for Request Handlers

```

class ExportStrategy(ABC):
    """Interface for exporting product data."""

    @abstractmethod
    def export(
        self,
        date: str,
        version: bool,
        data: List[Dict[str, Any]] | Dict[str, Any],
        output_path: str,
    ) -> bool:
        """
        Method for exporting to be implemented by subclasses.
        """

```

4.3.3 Using the `cpme__api` client

The tool utilizes an API client that wraps all interactions with the CPME API, making requests and handling responses. Our request handlers use respective methods for GET and POST requests, which return a JSON object back for further processing. This client also simplifies request construction for us.

4.3.4 Error handling and logging

To ensure that the user gets informed about errors in data or request, or alternatively about the progress of the request, we implement error handling and logging.

⁵https://github.com/ky573/CloudMarginEstimatorClient/blob/gk101_estimator_demo_tool_update/margin_estimator_tool/src/export_strategy/export_strategy.py

Error Handling

The tool handles API errors, invalid user inputs, and unexpected failures by:

- Using Python exceptions to catch errors and provide meaningful feedback to users.
- Doing retries in case of time-out for the request. This is handled by the `cpme_api` client.
- Validating user input before making API calls to prevent incorrect requests, for example validating filters or portfolio headers.

Logging

Instead of a logging framework, we use `click.echo` statements to keep the user informed about request progress and key stages of execution, like successful request or path to exported data. Information about the request is logged with the help of `cpme_api` client library, where the user gets informed about the attributes of the request.

Listing 4.3 Example of console output, HEADER truncated for clarity

```
2025-03-06 15:14:39,229 api-client INFO    cpme_api
configuration.enable_logging  Logger start.
2025-03-06 15:14:39,233 api-client INFO    cpme_api rest.GET
                        /products  HEADER:{'Content-Type':
                        'application/json', 'X-DBP-APIKEY': ...
Request successful.
Filtered down to 4748 items.
Products exported to /project root directory/
```

4.3.5 Environment variables

API keys are not hardcoded but instead stored in environment variables in a `.env` file. The tool uses the `os.getenv()` method to retrieve environment variables at runtime.

Additionally, if a proxy server is required to access the API, it can also be stored in the `.env` file. A proxy might be necessary in environments with restricted internet access, such as corporate networks or VPN setups.

4.3.6 Extending the application layer

The application layer is designed to be modular and extensible. If we want to extend it and add support for a new endpoint, we need to:

1. Define a class that implements the `RequestHandler` interface.
2. Implement the `process_and_provide_output` method from the base class to fetch data, optionally apply filters, and export it.
3. Register the handler in the endpoint handler factory to integrate it with the CLI.

4.3.7 Application layer flow

Figure 4.2 illustrates the request processing flow in the application layer. It is once again described using the UML flow diagram.

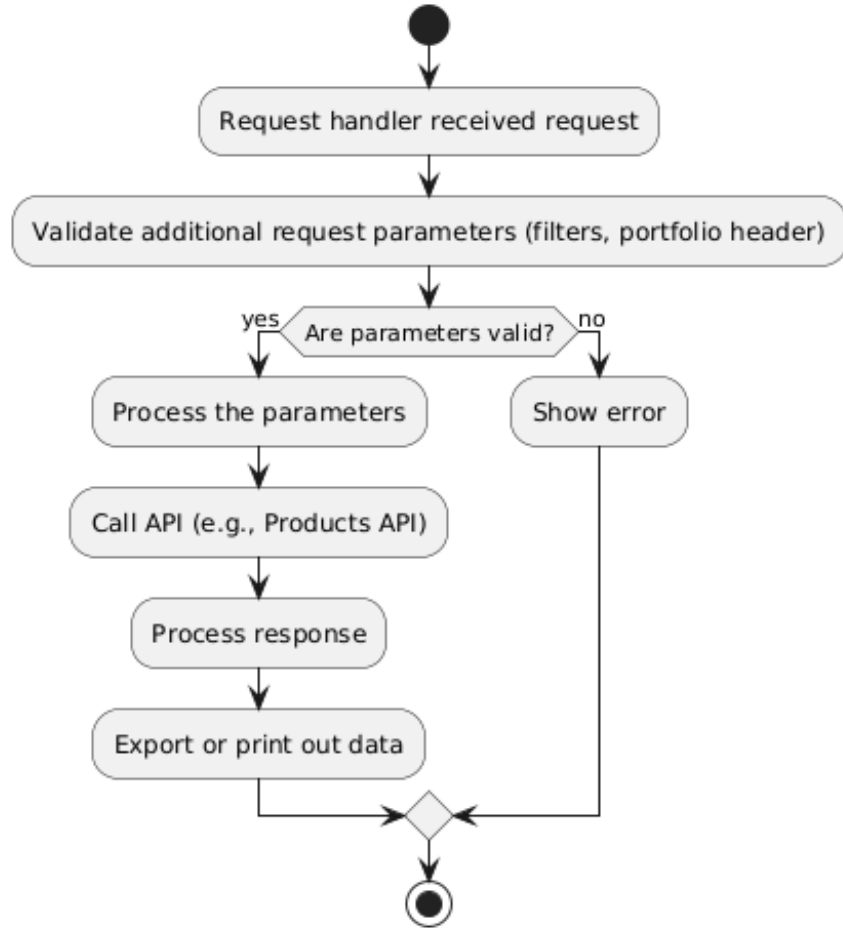


Figure 4.2 Application layer processing flow

4.4 Tool structure

After discussing the implementation of both CLI and the application layer, let us now lay out the structure. The source code of the project is structured as follows:

- cli/ - Manages command-line interface logic.
- core/ - Contains utilities and shared components.
- export_strategy/ - Implements different export strategies using a strategy pattern.
- estimator/ - Handles all operations related to margin estimation and estimator endpoint. It includes logic for margin calculation and portfolio upload.

- `live_snapshots/`, `products/`, `series/`, `snapshots/` - Dedicated request handlers for specific data types.
- `main.py` - The CLI application's entry point.

5 Testing

As was said in previous chapter, we use Pytest framework for testing. We also use some features from python built-in testing module unittest, like mocking and patching. Both the CLI and the application layer are tested by unit tests.

The tests are structured into directories based on the structure of the source code. Both public and private methods are tested. The test suites are automated and can be executed from the directory `margin_estimator_tool` by running:

```
pytest ./test
```

To be able to run the `pytest` command, the tool needs to be installed with developer dependencies. We will talk about the installation in chapter 6.

5.1 Test coverage

We can measure test coverage using the `pytest-cov` plugin. The coverage report can be run from `margin_estimator_tool` and generated using:

```
pytest --cov=src
```

The current results are presented in Table 5.1. Although coverage percentage provides a helpful overview of how much of the codebase is exercised by tests, it should not be considered as an indicator of test quality or completeness. The module we decided not to test is `command.py` file, as it covers configuring the commands and adding options to them, and the logic was tested in other tests. The other module we did not test is `main.py`, as it contains trivial logic.

Metric	Total Statements	Missed Statements	Coverage
Test Coverage	843	144	83%

Table 5.1 Summary of Python test coverage

5.2 Mocking

Mocking is used to isolate components during testing by replacing dependencies with controlled substitutes. This ensures that tests are reliable, fast, and independent of external factors. ¹

We use `unittest.mock` to replace API calls and other dependencies with mock implementations. Other techniques we use are:

- Patching: Temporarily replacing functions or objects during test execution.
- Mocking API Calls: Simulating external requests with predefined responses.
- MagicMock: Creating flexible mock objects to mimic dependencies.
- Asserting Calls: Ensuring that expected functions are called with correct parameters.

¹<https://www.telerik.com/products/mocking/unit-testing.aspx>

5.3 User testing

The application was internally tested by consultants at Deutsche Boerse Group. Issues and discrepancies in implementation were pointed out and subsequently resolved.

6 User documentation

In this section, we describe how to use the tool. We provide step-by-step instructions on how to install and configure the tool so that users can set up their environment correctly before executing commands. Furthermore, we describe how to use all of its commands. Each command will be presented with a description, including its purpose, required and optional parameters, and links to tutorials on GitHub for practical usage examples.

6.1 Installation and setup

Now, let us go through the installation and setup process, a setup of the `.env` file, and setup of Jupyter Notebook for the tutorials usage.

6.1.1 Installation

We need three prerequisites for our development:

- Python 3.10+
- Git ¹
- pip package manager

The whole project is in the git repository `CloudMarginEstimatorClient`, on the separate branch `gk101_estimator_demo_tool_update`. Our tool is located in the directory `margin_estimator_tool`.

Once we have opened the command prompt ² on a Windows system, or the terminal on a macOS or Linux system, we proceed with these steps for installation:

1. Clone the repository:

```
git clone https://github.com/ky573/CloudMarginEstimatorClient.git
cd CloudMarginEstimatorClient
```

2. Switch to the branch:

```
git switch gk101_estimator_demo_tool_update
```

3. Now, the `margin_estimator_tool` is visible on the branch

```
cd margin_estimator_tool
```

4. Create a virtual environment³:

```
python -m venv venv
```

5. Activate the virtual environment:

¹<https://git-scm.com/>

²https://tutorial.djangogirls.org/en/intro_to_command_line/

³<https://docs.python.org/3/library/venv.html>

- On Windows:

```
venv\Scripts\activate
```

- On macOS/Linux:

```
source venv/bin/activate
```

6. Install dependencies:

- For basic usage:

```
pip install .
```

- For development purposes, which includes tools for testing and linting:

```
pip install .[dev]
```

7. We also need to install cpme_api client:

```
cd .. # Go to the root project directory
pip install .
```

To check that the installation is successful, we can run the tool with the `--version` option:

```
margin_estimator_tool --version
```

The output should be:

```
margin_estimator_tool, version 1.0
```

6.1.2 Setup of the .env file

The tool also needs a `.env` file for storing the environment variables. The `.env` stores two values: the API key and the proxy.

The API key is free and can be obtained from DBG Digital Business Platform after signing into the company's Digital Platform. For our purposes, we want to choose Prisma Margin Estimator API key with a dedicated static IP. To obtain the key, we can follow the instructions on the website.

It may take a few minutes for the key to be functional. If the key is not yet functional, then the response contains, besides other information, also the following message, and the user should wait a few more minutes:

Error: (401)

Reason: Unauthorized HTTP response headers: HTTPHeaderDict({'Content-Security-Policy': 'default-src 'none'; frame-ancestors 'none'", 'Strict-Transport-Security': 'max-age=31536000; includeSubDomains; preload', ...

To be sure if the correct `.env` file is being loaded, the information about which API key is being used is available in the logging message:

```
2025-05-04 19:17:18,876 api-client INFO    cpme_api configuration ...
2025-05-04 19:17:18,887 api-client INFO    cpme_api rest.GET ...
HEADER: {'Content-Type': 'application/json', 'X-DBP-APIKEY': '88a3f90b-
?...'
```

The proxy address for requests might not be needed, as it is usually used only to bypass network restrictions in some company networks. Users can try the setup without the proxy first, and only in a situation when they get a time-out from a request, they can set the proxy. On Linux machines, one's proxy can be obtained by typing a command `http_proxy` into the terminal.

This is an example of a `.env` file:

```
API_KEY="1234abcd-12ab-12ab-12ab-abcdef123456"  
PROXY="http://proxy.company.cloud.eu:1234"
```

Without the proxy, it can look like this:

```
API_KEY="1234abcd-12ab-12ab-12ab-abcdef123456"
```

The `.env` file should be put into the `margin_estimator_tool` directory.

6.1.3 Using Jupyter Notebook for tutorials

For the purpose of tutorials and experimenting with the `margin_estimator_tool` in an interactive and user-friendly way, we use **Jupyter Notebook**. This allows us to run the tool and explore API responses step-by-step in a browser-based interface.

We can install Jupyter Notebook via `pip` by using the command:

```
pip install notebook
```

In the repository, we can find pre-prepared tutorial notebooks located in the `margin_estimator_tool/tutorials` directory. These notebooks contain usage examples, step-by-step guides, and instructions for working with the `margin_estimator_tool`.

To start using them:

1. Navigate to the `margin_estimator_tool` directory in the terminal.
2. Launch the Jupyter Notebook server:

```
python -m jupyter notebook
```

3. Open the `tutorials` directory in the browser interface.
4. Select the desired notebook and follow the instructions in the cells.

6.2 Showing all available commands

After successfully installing our tool, let us now see its capabilities. Our tool is available under the command `margin_estimator_tool`. We can see all of its commands by adding the argument `--help`:

```
margin_estimator_tool --help
```

We can see the output:

```
Usage: main.py [OPTIONS] COMMAND [ARGS]...
```

```
Main CLI group for margin estimator tool.
```

Options:

```
--version Show the version and exit.  
--help Show this message and exit.
```

Commands:

```
etd_portfolio      Uploads an ETD portfolio from a CSV file and...  
get_live_snapshots Fetch live snapshots and display information.  
get_products       Fetch products from the products endpoint.  
get_series         Fetch series from the series endpoint.  
get_snapshots      Fetch EOD snapshots (non-live) for the specified...  
margin_calculator Run estimator endpoint and calculate margins
```

6.3 Fetching available products

To get a list of available products, the user can use the `get_products` command. This command queries the products endpoint and returns available data about products for a given day.

The user can see all available options by adding argument `--help` to the command:

```
margin_estimator_tool get_products --help
```

It supports these options:

- Date: Specify the business date for which the data should be retrieved, it defaults to the current date.
- Version: Choose between EOD (end-of-day) and LIVE, it defaults to EOD in request.
- Timestamp: For LIVE version, optionally specify a timestamp to get a precise snapshot.
- Export format: Choose between CSV, which is default without specifying, Excel, or JSON using flags `--to_json` or `--to_excel`.
- Filters: Apply various filters to narrow down the selection of products.
- Export directory: Set a directory where the output file will be saved. It defaults to the current working directory.

Detailed usage can be found in tutorial [here](#).

6.4 Fetching series for products

To get a list of available series for specified products, the user can use the `get_series` command. The command queries the series endpoint and returns available data about products for a given day.

The user can see all available options by adding argument `--help` to the command:

```
margin_estimator_tool get_series --help
```

It supports these options:

- Date: Specify the business date for which the data should be retrieved, it defaults to the current date.
- Version: Choose between `EOD` (end-of-day) and `LIVE`, it defaults to `EOD` in request.
- Timestamp: For `LIVE` version, optionally specify a timestamp to get a precise snapshot.
- Export format: Choose between `CSV`, which is default without specifying, `Excel`, or `JSON` using flags `--to_json` or `--to_excel`.
- Export directory: Set a directory where the output file will be saved. It defaults to the current working directory.
- Products: Filter series by product names. This option is required.
- Type: Filter by instrument type — either `option` or `future`.
- Call/Put flag: Filter options by type — `C` for Call or `P` for Put.
- Template: Generate a template for an ETD portfolio.
- Max TTE: Filter series by maximum time to expiry in days.
- Min TTE: Filter series by minimum time to expiry in days.
- Generic filter: Provide comma-separated key:value pairs to filter based on specific series fields.

Detailed usage can be found in tutorial [here](#).

6.5 Fetching live snapshots

To retrieve data about available live snapshots for the given business day, the user can use the `get_live_snapshots` command. The command prints out the data to the command line in a formatted output.

The user can see all available options by adding argument `--help` to the command:

```
margin_estimator_tool get_live_snapshots --help
```

This command supports only a single option `-date`, which is required. It specifies the date for live snapshots.

Detailed usage can be found in tutorial [here](#).

6.6 Fetching snapshots

Data of end-of-day or first live snapshots are available to the user after using the `get_snapshots` command. The command prints out the data for the given range of dates to the command line in a formatted output.

The user can see all available options by adding the argument `--help` to the command:

```
margin_estimator_tool get_snapshots --help
```

For this command, the user can specify two options. The first of the options is `--date_from`, which is required. The second option is `--date_to`, which defaults to the current date.

Detailed usage can be found in tutorial [here](#).

6.7 Sending a portfolio to the estimator

To send an ETD portfolio to the estimator, the user can use the `etd_portfolio` command. This command processes the portfolio from a CSV file, sends it to the estimator API, and returns the calculated margin results.

The user can see all available options by adding the argument `--help` to the command:

```
margin_estimator_tool etd_portfolio --help
```

It supports these options:

- CSV file: Provide the path to the ETD portfolio file in CSV format. This option is required.
- Date: Specify the business date in YYYYMMDD format. If not provided, it defaults to the current date.
- Version: Choose between EOD (end-of-day) and LIVE, it defaults to EOD in request.
- Timestamp: For the LIVE version, optionally provide a specific timestamp to retrieve an exact snapshot.
- Export format: Choose the output format of the results — by default, results are printed to the console, but they can be exported to Excel or JSON using the flags `--to_excel` or `--to_json`.
- Export directory: Specify the directory where the output file should be saved using `--export_dir`. Defaults to the current working directory.

An example portfolio might look like this:

```
Product ID,Contract Date,Call Put Flag,Exercise Price,Version Number,  
Net LS Balance  
FEXD,20311219,,0,100  
OEXD,20311219,C,100.00,0,-100
```

Listing 6.1 Example ETD portfolio in CSV format⁴

Detailed usage can be found in tutorial [here](#).

6.8 Calculating and aggregating the margins

To calculate and aggregate the margins by sending ETD portfolio to the estimator endpoint, the user can use the `margin_calculator` command. This command processes the portfolio from the CSV file and sends it for an estimation to the estimator endpoint for each business date in the specified interval. Then it aggregates the requests, and exports initial margins in the form of a graph and in Excel form.

The user can see all available options by adding the argument `--help` to the command:

```
margin_estimator_tool margin_calculator --help
```

It supports these options:

- CSV file: Provide the path to the ETD portfolio file in CSV format. This option is required.
- Start Date: Specify the business date for the start of interval in YYYYMMDD format. This option is required.
- End Date: Specify the business date for the end of interval in YYYYMMDD format. This option is required.
- Version: Choose between EOD (end-of-day) and LIVE, it defaults to EOD in request.
- Timestamp: For the LIVE version, optionally provide a specific timestamp to retrieve an exact snapshot.
- Export directory: Specify the directory where the output file should be saved using `--export_dir`. This option is required.

Detailed usage can be found in tutorial [here](#).

6.9 Limitations and unsupported features

This section outlines the current limitations of our CLI tool and features that are not supported.

6.9.1 OTC portfolio support

The tool currently supports only ETD portfolios and does not support OTC portfolios. As a result, any request involving OTC portfolios will not be processed correctly. Users must ensure that only ETD portfolios are used when making API requests.

⁴Source: <https://deutsche-boerse-risk.github.io/CloudPrismaMarginEstimator/docs/gui.html>

6.9.2 Historic data request delays

Requests for historic dates can experience significant delays due to external API limitations. The API used for retrieving historic data may take up to 5 minutes to process a single request. This delay is outside the control of the CLI tool, as it depends on the API provider's infrastructure. Users should plan accordingly when making historical data queries.

6.10 Tutorials

As was mentioned in the subsection 2.2.4, a set of tutorials has been prepared to help users understand how to work with the tool in practice. These tutorials are provided as Jupyter Notebook files and are available in the GitHub repository under the `margin_estimator_tool/tutorials` directory.

The available tutorials are:

- `get_products_tutorial.ipynb` – How to fetch available products.
- `get_series_tutorial.ipynb` – How to fetch series for products.
- `live_snapshots_tutorial.ipynb` – How to fetch live snapshots.
- `snapshots_tutorial.ipynb` – How to fetch EOD snapshots.
- `etd_portfolio_tutorial.ipynb` – How to upload a portfolio to the estimator.
- `margin_calculator_tutorial.ipynb` – How to calculate and aggregate margins.
- `calendar_spread_tutorial.ipynb` – Example of a calendar spread strategy.

The tutorials are not fully included in the thesis for brevity.

7 Use case - calendar spread strategy

After providing the user documentation in the previous chapter, we now discuss a calendar spread strategy with regard to our tool. First, we introduce the key terms associated with the strategy. Then, we introduce the strategy itself and how margins relate to the strategy. Finally, we provide users with a tutorial on how to implement it using our tool.

7.1 Key terms

Now, let us define some key terms to help us understand the strategy.

Intrinsic value

The intrinsic value of an option measures the immediate profit that could be gained if the option were exercised at the current moment. For a call option, this is the amount by which the underlying asset's price exceeds the strike price; for a put option, it is the amount by which the strike price exceeds the underlying asset's price.¹ Options that have a positive intrinsic value are called in-the-money, whereas those with no intrinsic value are out-of-the-money. Intrinsic value excludes any consideration of time value or future price movements.²

Time decay

Time decay, also called theta, describes the reduction in the price of an option as it approaches its expiration date. [22] An option's value consists of its intrinsic value and also of its time value — the possibility that the underlying asset's price will move in a favorable direction before expiration. As time passes, the possibility of moving in favorable direction decreases. This causes the time value component of the option's price to decline. The time decay accelerates as the expiration date of the option approaches. This means that short-term³ options lose their time value more quickly than longer-term⁴ options. [23]

Premium

The premium of an option is the total price paid by the buyer to acquire the option from the seller. It reflects both the intrinsic value and the time value of the option. The premium might be greater for an option with more time to expiration or greater implied volatility. [24]

¹<https://www.merrilledge.com/investment-products/options/options-pricing-valuation>

²<https://www.ig.com/en/glossary-trading-terms/intrinsic-value-definition>

³An option contract that has an expiration date relatively close to the current date, typically in a range of a few weeks

⁴An option contract that has an expiration date that extends beyond that of the shorter-term option, usually around a year, but can be also shorter

Implied volatility

Implied volatility represents the market's expectations for the future volatility of the underlying asset over the life of the option. Higher demand or rising expectations increase implied volatility and option premiums, while lower demand or falling expectations decrease both. Changes in implied volatility affect the time value of options and can impact trade outcomes, even if the stock moves as predicted. Long-dated options are more sensitive to volatility changes than short-dated ones due to their greater time value. Higher implied volatility increases the option's premium because the likelihood of large price movements — and therefore the chances that the option will become profitable — is greater. [25] Because of that, implied volatility plays a crucial role in the pricing of options and impacts strategies that rely on the relationship between different expiration dates.

Net debit

A net debit refers to a situation where the total amount paid for entering an options strategy with multiple legs ⁵ exceeds the amount received. In a long calendar spread, the trader buys a longer-term option and simultaneously sells a shorter-term option. Because the longer-term option is generally more expensive due to its greater time value, the result is a net debit — the trader must pay an initial cost to establish the position. This initial investment also defines the maximum potential loss of the strategy. [26]

7.2 Definition of a calendar spread

A calendar spread is an options trading strategy that involves the simultaneous sale of a shorter-term option and the purchase of a longer-term option of the same type, which can be call or put, and with the same strike price, all based on the same underlying asset. [27] It is also frequently referred to as a horizontal spread, as it is characterized by buying options with the same strike price and different expiration dates.

There are two main types of calendar spread, the long calendar spread and the short calendar spread. The long calendar spread is established by buying a long-term option and selling a short-term option, whereas in the short calendar spread, an investor does the opposite - selling a long-term option and buying a short-term one. In our case, we focus only on the long calendar spread.

7.2.1 Long calendar spreads

Long calendar spreads are advantageous when the price of an underlying asset remains relatively stable or trades within a narrow range in the near term, especially around the strike price of the spread. [27] In such a scenario, profit occurs due to the difference in the rate of time decay between the shorter-term and longer-term options. The short-term option's time value declines faster than the long-term option. If the price of the underlying asset remains near the strike price, then the short-term options can expire worthless. An option expires worthless

⁵Multiple legs means multiple options

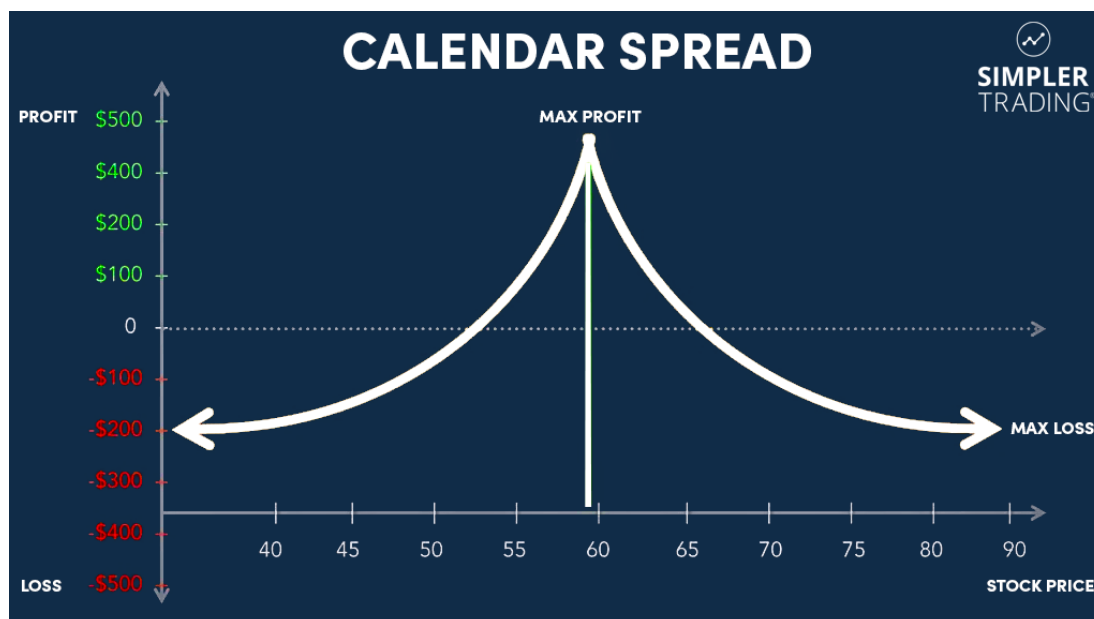


Figure 7.1 Profit of a calendar spread. Taken from [28]

when it reaches its expiration date and has no intrinsic value. When the short-term option expires worthless, it allows the trader to profit from the premium which he receives from selling it, as he gets to keep all that money without any cost. This can help to offset the initial cost of the long-term option.

Figure 7.1 shows the profit and loss profile of a long calendar spread strategy. The horizontal axis represents the underlying stock price at the expiration of the short-term option, while the vertical axis shows the profit or loss associated with it. In our illustration, the calendar spread is constructed with a strike price of \$59. The strategy achieves its maximum profit, approximately \$500 when the stock price at the expiration of the short-term option is exactly at the strike price of \$59.

The profit reached from the spread declines as the price of the underlying stock deviates from the strike price in either direction. The stock prices of \$53 and \$65 are the points of no profit, but also no loss, thus the trader breaks even. If the stock price moves significantly, falling either towards \$40 or rising towards \$80, the trader reaches the maximum loss of approximately -\$200.

This can be explained by the time decay. If the stock price stays close to the strike price, the short option expires worthless while the long-term option retains significant value, maximizing the trader's profit. On the other hand, big movements away from the strike price reduce the value of both options, resulting in a net loss limited to the initial cost (net debit) of establishing the spread.

One of the advantages of a long calendar spread is its defined risk profile. The maximum potential loss for this strategy is typically limited to the net debit collected when establishing the spread.⁶ This loss occurs when both the short-term and the long-term options expire worthless. By selling the short-term option, the trader receives a premium that helps to offset the cost of buying the long-term option. This increases the capital efficiency of the strategy compared to only buying the long-term option, as it requires a smaller initial investment.

⁶<https://www.optionsplaybook.com/option-strategies/calendar-call-spread>

7.3 Margin and calendar spreads

As we said earlier, when trading options, margin refers to the collateral an investor must provide to cover potential obligations. Unlike margin in stock and futures trading, which is primarily used to leverage positions and increase buying power, margin in options trading mainly functions as a risk management tool for brokers when clients undertake the obligation of selling options. [29] Buying options usually only requires the payment of the premium, as the risk of the buyer is limited to this amount. However, when a position includes short options, brokers demand a margin to ensure the seller can meet potential losses if assigned.

7.3.1 Margin treatment in calendar spreads

Calendar spreads can influence margin requirements through their structure. The long option in the spread acts as a hedge for the short option, which can lower the required margin compared to holding an uncovered, so-called naked short option.⁷ The long option acts to limit the potential for losses beyond a certain level. Specifically, for long calendar spreads, which are established with a net debit, meaning an initial cost to the trader, the margin requirement is frequently linked to the net cost of establishing the spread.⁸ The maximum potential loss for a long calendar spread is the initial debit paid, and this defined level of risk can result in more favorable margin treatment from brokerage firms.

7.3.2 Assignment risk and margin changes

As the short-term leg of a calendar spread nears expiration, the risk of early assignment increases, especially if the option moves in-the-money. An in-the-money option means its strike price is favorable compared to the current market price. For calls, the stock price is above the strike, and for puts, the stock price is below the strike. This makes it more likely the holder will choose to exercise early.⁹ If a trader is assigned a short option, they become obligated to either buy or sell the underlying asset at the option's strike price. The longer-term option within the calendar spread provides a certain degree of protection against this assignment risk, as the trader has the right to exercise their long option to potentially fulfil the obligation arising from the short option assignment. This is because both options share the same strike price. Once the short option expires, the spread basically becomes a single long option position. Since these options typically do not have margin requirements beyond the initial premium paid, the margin associated with the short-term option would be removed from the account.

7.3.3 Summary of margin benefits

In conclusion, compared to other strategies, calendar spreads, particularly long ones, provide more predictable and often lower margin requirements than positions involving naked short options, thanks to their risk protection.

⁷<https://www.schwab.com/learn/story/how-could-vertical-spreads-help-your-strategy>

⁸<https://www.optionsplaybook.com/option-strategies/calendar-call-spread>

⁹<https://www.investopedia.com/ask/answers/042715/what-difference-between-money-and-out-mo>
asp

7.4 Tutorial for calendar spread

After defining what a calendar spread is and defining its relation to margins, we will now take a look at how it can be implemented and analyzed using our tool. A detailed tutorial can be found [here](#).

Conclusion

In this thesis, we have dealt with developing the command-line interface tool for Cloud Prisma Margin Estimator. We started by introducing the estimator and showcasing its current usages, together with their respective drawbacks and advantages. This helped us define the requirements for our new tool, which we analyzed in the next chapter. After that, we created the design of the application and described its implementation. In the last part of the thesis, we provided the user documentation together with the tutorials and the description of the use case for the Calendar spread strategy.

Collaboration with Deutsche Boerse Group was very professional. There were regular meetings throughout the whole process of the writing of the thesis on a bi-weekly basis, and the dedicated company representatives were always available. We successfully defined and shaped the requirements together and consulted on the progress throughout the year.

The tool also has some areas for expansion. More endpoints could be supported, or more types of exports could be added. The `cpme_api` client can also be improved. Currently, it serves the requests synchronously, but in the future, an asynchronous implementation could be developed to improve the performance.

In conclusion, the result of our thesis is a fully functional command-line interface tool that interacts with Cloud Prisma Margin Estimator. This tool meets the required criteria stated by the company and is ready to be used by its clients.

Bibliography

1. HAYES, Adam. *Stocks: What They Are, Main Types, How They Differ From Bonds*. 2024. Available also from: <https://www.investopedia.com/terms/s/stock.asp>. Accessed: 2025-03-15.
2. FERNANDO, Jason. *Bonds: How They Work and How To Invest*. 2024. Available also from: <https://www.investopedia.com/terms/b/bond.asp>. Accessed: 2025-03-15.
3. HAYES, Adam. *Understanding Derivatives: A Comprehensive Guide to Their Uses and Benefits*. 2025. Available also from: [https://www.investopedia.com/terms/d/derivative.asp#:~:text=The%20term%20%E2%80%9Cderivative%E2%80%9D%20refers%20to%20,the%20counter%20\(OTC\)..](https://www.investopedia.com/terms/d/derivative.asp#:~:text=The%20term%20%E2%80%9Cderivative%E2%80%9D%20refers%20to%20,the%20counter%20(OTC)..) Accessed: 2025-03-15.
4. HAYES, Adam. *Options vs. Futures: What's the Difference?* 2024. Available also from: <https://www.investopedia.com/ask/answers/difference-between-options-and-futures/>. Accessed: 2025-04-05.
5. TREFETHEN, John. *Five Differences Between OTC and Exchange Traded Derivatives*. 2024. Available also from: <https://hedgestar.com/single-post/2024/06/27/five-differences-between-otc-and-exchange-traded-derivatives/#:~:text=OTC%20derivatives%20are%20traded%20directly,is%20facilitated%20by%20the%20exchange..> Accessed: 2025-03-05.
6. FERNANDO, Jason. *Margin and Margin Trading Explained Plus Advantages and Disadvantages*. 2024. Available also from: <https://www.investopedia.com/terms/m/margin.asp>. Accessed: 2025-03-05.
7. CHEN, James. *Margin Call: What It Is and How to Meet One With Examples*. 2024. Available also from: <https://www.investopedia.com/terms/m/margincall.asp>. Accessed: 2025-03-05.
8. GOCARDLESS. *How Do Hedging Strategies Work?* 2021. Available also from: <https://gocardless.com/guides/posts/how-do-hedging-strategies-work/#:~:text=The%20hedging%20meaning%20in%20finance,the%20impact%20of%20unexpected%20risk..> Accessed: 2025-03-16.
9. TINCHER, Laurel. *What Is Portfolio Margin?* 2024. Available also from: <https://www.sofi.com/learn/content/what-is-portfolio-margin/>. Accessed: 2025-04-06.
10. HAYES, Adam. *Portfolio Margin: Overview, How it Works*. 2022. Available also from: <https://www.investopedia.com/terms/p/portfolio-margin.asp>. Accessed: 2025-03-05.
11. CHEN, James. *Interest Rate Swap: Definition, Types, and Real-World Example*. 2024. Available also from: <https://www.investopedia.com/terms/i/interestrateswap.asp>. Accessed: 2025-04-06.

12. TEAM, AltexSoft Editorial. *Functional and Nonfunctional Requirements: Specification and Types*. 2023. Available also from: <https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/>. Accessed: 2025-03-18.
13. MARTIN, R. *Clean Architecture*. Pearson Education, Inc., 2018. First Edition. ISBN 0-13-449416-4.
14. PUNUGUPATI, Venkatesu. *Software Architecture Design Principles*. 2022. Available also from: <https://www.linkedin.com/pulse/software-architecture-design-principles-venkatesu-punugupati/>. Accessed: 2025-03-24.
15. NECASKY, Martin. *Architecture styles - monolithic*. 2024. Available also from: https://cunicz-my.sharepoint.com/:p:/r/personal/11165798_cuni_cz/_layouts/15/Doc.aspx?sourcedoc=%7B9EC6E1C9-A40B-4B97-950A-2E2B63C498E0%7D&file=NSWI130_06_L_styles_monolithic.pptx&action=edit&mobileredirect=true. Accessed: 2025-04-28.
16. SANDARUWAN, Dulanjaya. *Day 2: The Layered Architecture Pattern*. 2024. Available also from: <https://medium.com/@dulanjayasandaruwan1998/day-2-the-layered-architecture-pattern-60e99ab15059>. Accessed: 2025-03-24.
17. JOVANOVIC, Milan. *What Is a Modular Monolith?* 2024. Available also from: <https://www.milanjovanovic.tech/blog/what-is-a-modular-monolith>. Accessed: 2025-04-07.
18. GAMMA, E. et al. *Design Patterns : Elements of Reusable Object-Oriented Software*. Reading, Mass. : Addison-Wesley, 1995. First Edition. ISBN 0-201-63361-2.
19. MARINACCI, Josh. *What is a Code Formatter and why would I want to use one?* 2024. Available also from: <https://trunk.io/learn/what-is-a-code-formatter-and-why-would-i-want-to-use-one>. Accessed: 2025-03-28.
20. BULEJ, Lubormir. *Design Patterns meet Design Principles*. 2025. Available also from: <https://d3s.mff.cuni.cz/f/teaching/nprg043/06-patterns.pdf>. Accessed: 2025-04-28.
21. FREEMAN, E.; ROBSON, E. *Head First Design Patterns*. O'Reilly Media, Inc., 2021. Second Edition. ISBN 978-1-492-07800-5.
22. JAISWAL, Tanushree. *Time Decay*. 2025. Available also from: <https://www.5paisa.com/blog/what-is-time-decay-in-option-trading#:~:text=Time%20decay%2C%20also%20known%20as,the%20underlying%20asset's%20price%20movement..> Accessed: 2025-04-16.
23. CHEN, James. *What Is Time Decay? How It Works, Impact, and Example*. 2024. Available also from: <https://www.investopedia.com/terms/t/timedecay.asp>. Accessed: 2025-04-28.
24. KENTON, Will. *Option Premium: Definition, Factors Affecting Pricing, and Example*. 2022. Available also from: <https://www.investopedia.com/terms/o/option-premium.asp>. Accessed: 2025-04-28.

25. KOHLER, Jeff. *Implied Volatility: Buy Low and Sell High*. 2024. Available also from: <https://www.investopedia.com/articles/optioninvestor/08/implied-volatility.asp#:~:text=Implied%20volatility%20is%20the%20expected,to%20changes%20in%20implied%20volatility..> Accessed: 2025-04-28.
26. CHEN, James. *Calendar Spreads in Futures and Options Trading Explained*. 2025.
27. IDO, Satoshi. *The Calendar Spread Options Strategy (and How to Build with Alpaca)*. 2025. Available also from: <https://alpaca.markets/learn/calendar-spread>. Accessed: 2025-04-16.
28. HUELSTER, Cody. *Everything You Need to Know about Calendar Spreads*. 2022. Available also from: <https://www.simplertrading.com/blog/calendar-spreads>. Accessed: 2025-04-28.
29. CHEN, James. *Option Margin: Definition, Requirements, How To Calculate*. 2022. Available also from: <https://www.investopedia.com/terms/o/option-margin.asp>. Accessed: 2025-04-17.

List of Figures

1.1	GUI on the web	13
3.1	Layered Architecture[16]	22
3.2	C4 model - Software system layer	24
3.3	C4 model - Container layer	24
3.4	C4 model - CLI components	25
3.5	C4 model - Application layer components	28
4.1	CLI processing flow	32
4.2	Application layer processing flow	35
7.1	Profit of a calendar spread. Taken from [28]	49

List of Tables

1.1	Comparison of CPME Access Methods	15
5.1	Summary of Python test coverage	37